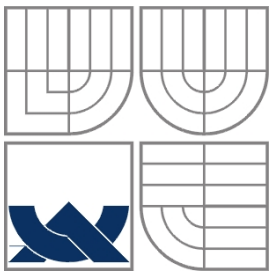
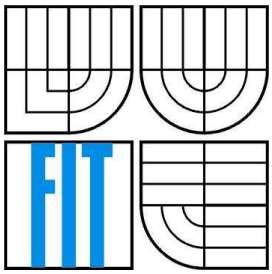


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

UNIVERZÁLNÍ TABULKOVÝ EDITOR V PHP

UNIVERSAL WEB DATAGRID EDITOR IN PHP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. EMIL FRÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROMÍR MARUŠINEC, Ph.D.

BRNO 2008

Abstrakt

Cílem této diplomové práce je vytvoření univerzálního tabulkového editoru v PHP. Systém byl implementován za použití technologií HTML, PHP, Oracle a JavaScript.

Klíčová slova

Univerzální tabulkový editor, databáze, Oracle, HTML, PHP, JavaScript.

Abstract

The main topic of this master's thesis is to create a universal web datagrid editor. System was implemented using by HTML, PHP, Oracle and JavaScript.

Keywords

Universal web datagrid editor, database, Oracle, HTML, PHP, Javascript.

Citace

Fránek Emil: Univerzální tabulkový editor v PHP. Brno, 2008, diplomová práce, FIT VUT v Brně.

Univerzální tabulkový editor v PHP

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaromíra Marušince, Ph.D.
Další informace mi poskytl Ing. Marek Strakoš.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Na tomto místě bych chtěl poděkovat Ing. Marku Strakošovi za odborné vedení, užitečné rady a konzultace k mé diplomové práci.

Obsah

1 Úvod

Cílem této diplomové práce je vytvořit univerzální tabulkový editor. Jedná se o komponentu, která uživateli usnadní práci s databází Oracle. Komponenta bude vytvořena pomocí programovacího jazyka PHP. Bude tedy ovladatelná přes webový prohlížeč, takže i méně zkušeným uživatelům počítače bude umožňovat pracovat s daty uloženými v databázi Oracle. Těm zdatnějším zase usnadní a urychlí práci s touto databází.

Druhá kapitola této práce se zabývá databázemi. V první části je pozornost věnována obecné teorii databází, včetně nástinu postupného vývoje databázových systémů. Dále je popsáno několik základních typů databázových modelů, přičemž větší prostor je věnován databázím relačním, protože tyto databáze jsou dnes nejrozšířenější. U relačních databází se věnuji jejich historii, struktuře, integritním omezením a relační algebře, která nám umožňuje zpracování dat z databáze. Na konci této kapitoly rozebírám dotazovací jazyk SQL, jakožto prostředek pro práci s databázemi.

Třetí kapitola se zabývá analýzou portálu VUT. Protože není k této problematice dostatek dostupných podrobných informací, při vypracování jsem vycházel z diplomové práce Ing. Marka Strakoše [9], která se také zabývá tímto tématem. Na začátku této kapitoly jsou naznačeny požadavky na portál VUT, následně je krátce nastíněna struktura portálu s popisem nejdůležitějších částí. Nechybí krátký přehled používaného hardwaru, softwaru a důležitých knihoven. Tato kapitola dále objasňuje způsob provádění autentizace jednotlivých uživatelů a způsob organizace zapojení jednotlivých serverů portálu.

Čtvrtá kapitola pojednává o datagridech. Nejprve je zmíněna obecná charakteristika datagridu s uvedením, které základní funkce datagridy uživateli poskytují. Následuje výčet nejznámějších dostupných nástrojů pro práci s databázovými systémy. U těchto nástrojů je pro názornost uveden i obrázek, na kterém je zachycen jejich vzhled.

Další kapitola se týká návrhu vytvářené komponenty. Jsou zde uvedeny základní požadavky na funkčnost, včetně popisu návrhu jejich řešení. Z tohoto návrhu následně vychází implementace komponenty.

V páté kapitole se snažím rozebrat programovací jazyky a prostředí, které byly použity k implementaci tabulkového editoru. Protože vytvořená komponenta je ovládána pomocí webového prohlížeče, zabývám se jazyky HTML a PHP. Komponenta by měla co nejméně zatěžovat databázový server a komunikaci po síti, z tohoto důvodu použiji při implementaci i technologií JavaScriptu a DOM modelu. Je uveden i základní popis technologie AJAX, jakožto zastřešovací technologie pro JavaScript a DOM model. Dále je rozebrána problematika databázového systému Oracle, nad kterým bude celá komponenta vlastně pracovat. Druhá část kapitoly se týká vlastní implementace. Je zde podrobně uvedeno, jakým způsobem byly implementovány jednotlivé části komponenty. Nechybí obrázky demonstrující jednotlivé funkce.

V závěru jsou zhodnoceny dosažené výsledky této diplomové práce, včetně zhodnocení všech vlastností vytvořené komponenty.

2 Databáze

Databáze (neboli datová základna) je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. Tento systém se v české odborné literatuře nazývá systém řízení báze dat (SŘBD). Běžně se označením databáze – v závislosti na kontextu – myslí jak uložená data, tak i software (SŘBD). [1]

2.1 Historie databází

Předchůdcem elektronických databází, jak je známe v dnešní době, byly papírové kartotéky. Ty umožňovaly uspořádávání dat podle různých kritérií a zatřiďování nových položek. Veškeré operace s nimi se však musely provádět manuálně, vykonával je tedy přímo člověk. Správa takových kartoték byla v mnohém podobná správě dnešních databází.

Prvním krokem vývoje bylo převedení zpracování dat na stroje. Za první velké strojové zpracování dat lze asi považovat sčítání lidu ve Spojených státech v roce 1890. Paměťovým médiem byl tehdy děrný štítek a zpracování sebraných informací probíhalo na elektromechanických strojích. Elektromechanické stroje se využívaly pro účely zpracování dat další půlstoletí.

Velkým impulsem pro další rozvoj databází byl překotný vývoj počítačů v období padesátých let 20. století. Ukázalo se, že původně univerzální používání strojového kódu procesorů je (nejen) pro databázové úlohy příliš neefektivní, a proto se objevil nový požadavek na vyšší jazyk pro zpracování dat v databázích.

V roce 1959 se konala konference zástupců firem, uživatelů a amerického ministerstva obrany, jejímž závěrem byl požadavek na univerzální databázový jazyk. Výsledkem byla o rok později na konferenci CODASYL publikovaná první verze jazyka COBOL, který byl po mnoho dalších let nejrozšířenějším jazykem pro hromadné zpracování dat.

V roce 1965 na konferenci CODASYL byl vytvořen výbor Database Task Group (DBTG), který měl za úkol vytvořit koncepci databázových systémů. Začaly vznikat první síťové SŘBD na sálových počítačích. Jedním z prvních průkopníků databází byl Charles Bachman.

V roce 1971 vydal výbor zprávu The DBTG April 1971 Report, kde se objevily pojmy jako schéma databáze, jazyk pro definici schématu, subschéma a podobně. Byla zde popsána celá architektura síťového databázového systému.

Ve stejné době byly vyvíjeny i hierarchické databáze. Jedním z prvních SŘBD byl IMS, který byl vyvinut firmou IBM pro program letu na Měsíc - Program Apollo. Systém IMS patří stále k nejrozšířenějším na sálových počítačích.

V roce 1970 začínají zveřejněním článku E. F. Codda první relační databáze, které pohlížejí na data jako na tabulky. Kolem roku 1974 se vyvíjí první verze dotazovacího jazyka SQL. Vývoj této technologie po 10 letech přinesl výkonově použitelné systémy, srovnatelné se síťovými a hierarchickými databázemi.

V 90. letech 20. století se začínaly objevovat první objektově orientované databáze, jejichž filozofie byla přebírána z objektově orientovaných jazyků. Tyto databáze měly podle předpokladů vytlačit relační systémy. Původní předpoklady se však nenaplnily a vznikla kompromisní objektově-relační technologie. [1]

2.2 Databázové modely

Tato podkapitola se zabývá různými databázovými modely. Jsou zde uvedeny základní charakteristiky jednotlivých modelů a případné odlišnosti mezi těmito modely. Dnes se nejvíce používá model relační, proto mu bude věnována celá podkapitola.

Z hlediska způsobu ukládání dat a vazeb mezi nimi můžeme rozdělit databáze do několika základních typů:

- Hierarchická databáze
- Síťová databáze
- Relační databáze
- Objektová databáze
- Objektově relační databáze

Následují charakteristiky jednotlivých modelů.

2.2.1 Hierarchická databáze

Tato databáze je založená na hierarchickém modelu. Hierarchický model je speciálním typem síťového modelu omezující logické uspořádání dat na stromovou strukturu, jež umožňuje vyjádřit ve směru shora dolů jednosměrné vztahy typu 1 - více. Vztahy mezi záznamy jsou vyjádřeny obvykle prostřednictvím ukazatelů (pointerů), tj. speciálních položek obsahujících odkaz na identifikátor souvisejícího záznamu. Manipulace s daty se děje procedurálně, sekvenčním procházením stromem nebo jeho částí (větví). [2]

2.2.2 Síťová databáze

Jedná se o databázi založenou na síťovém modelu, ve kterém jsou data logicky i fyzicky uspořádána jako uzly rovinného grafu, v němž může být každý záznam spojený s libovolným počtem dalších záznamů. Vztahy mezi záznamy jsou vyjádřeny obvykle prostřednictvím ukazatelů (pointerů), tj.

speciálních položek obsahujících odkaz na identifikátor souvisejícího záznamu. Manipulace s daty se děje procedurálně procházením grafu cestou definovanou ukazateli, tzv. navigací. V praktických implementacích jsou síťové modely zpravidla omezeny - např. standard CODASYL připouští pouze vztahy 1 - více, v síti WWW nejsou (zatím) zavedeny obousměrné odkazy apod. [2]

2.2.3 Objektová databáze

Pro objektové databáze neexistuje žádný oficiální standard. Standardem je de facto kniha Morgana Kaufmana The Object Database Standard: ODMG-V2.0. Důraz se klade na přímou korespondenci mezi následujícími:

- objekty a objektové vztahy v aplikaci napsané v OO jazycích
- jejich uchovávání v databázi.[3]

2.2.3.1 Datový model

Objektové databáze využívají datového modelu, který má objektově orientované aspekty jako třídy s atributy, metodami a integritními omezeními. Poskytují objektové identifikátory (OID) pro každou trvalou instanci třídy. Podporují zapouzdření (encapsulation), násobnou dědičnost (multiple inheritance) a abstraktní datové typy.

Objektové databáze kombinují prvky objektově orientovaného programování s databázovými schopnostmi. Rozšiřují funkčnost objektových programovacích jazyků (C++, Smalltalk, Java) a poskytují plnou schopnost programování databáze. Datový model aplikace a datový model databáze se ve výsledku hodně shodují a výsledný kód se dá mnohem efektivněji udržovat. [3]

2.2.3.2 Dotazovací jazyk

Objektově orientovaný jazyk (C++, Java, Smalltalk) je jazykem jak pro aplikaci, tak i pro databázi. Poskytuje těsný vztah mezi objektem aplikace a uloženým objektem. Názorně je to vidět v definici a manipulaci s daty a v dotazech. [3]

2.2.3.3 Výpočetní model

V klasické relační databázi rozumíme dotazovacím jazykem vytváření, přístup a aktualizaci objektů, ale v objektové databázi, ačkoliv je to stále možné, je toto prováděno přímo pomocí objektového jazyka (C++, Java, Smalltalk) využitím jeho vlastní syntaxe. Navíc každý objekt v systému automaticky obdrží identifikátor (OID), který je jednoznačný a neměnný během existence objektu. [3]

2.2.4 Objektově relační databáze

"Rozšířená relační" a "objektově-relační" jsou synonyma pro databázové systémy, které se snaží sjednotit rysy jak relačních, tak objektových databází. Objektově relační databáze je specifikována v rozšíření SQL standardu — SQL3. Do této kategorie patří např. Informix, IBM, Oracle a Unisys. [3]

2.2.4.1 Datový model

Objektově relační databáze využívají datový model tak, že "přidávají objektovost do tabulek". Všechny trvalé informace jsou stále v tabulkách, ale některé položky mohou mít bohatší datovou strukturu, nazývanou abstraktní datové typy (ADT). ADT je datový typ, který vznikne zkombinováním základních datových typů. Podpora ADT je atraktivní, protože operace a funkce asociované s novými datovými typy mohou být použity k indexování, ukládání a získávání záznamů na základě obsahu nového datového typu. Objektově relační databáze jsou nadmnožinou relačních databází a pokud nevyužijeme žádné objektové rozšíření jsou ekvivalentní s SQL2. Proto mají omezenou podporu dědičnosti, polymorfismu, referencí a integrace s programovacím jazykem. [3]

2.2.4.2 Dotazovací jazyk

Objektově relační databáze podporují rozšířenou verzi SQL — SQL3. Důvodem je podpora objektů (tj. dotazy obsahující atributy objektů). Typická rozšíření zahrnují dotazy obsahující vnořené objekty, atributy, abstraktní datové typy a použití metod. Objektově relační databáze jsou stále relační, protože data jsou uložena v řádcích a sloupcích tabulek a SQL, včetně zmíněných rozšíření, pracuje právě s nimi. [3]

2.2.4.3 Výpočetní model

Jazyk SQL s rozšířením pro přístup k ADT je stále hlavním rozhraním pro práci s databází. Přímá podpora objektových jazyků stále chybí, což nutí programátory k překladu mezi objekty a tabulkami.[3]

2.3 Relační databáze

Jak již bylo dříve uvedeno, jedná se o dnes nejrozšířenější databáze, proto jim budu věnovat i více pozornosti.

2.3.1 Historie relačních databází

Základem relačních databázových systémů se stala publikace pracovníka firmy IBM E.F.Codda s názvem „A relational data model for large shared data banks“, která vyšla v roce 1970 v časopisu Communications of the ACM. V této publikaci byl zaveden zcela nový datový model pro ukládání perzistentních dat s cílem dosažení datové nezávislosti, tj. nezávislosti aplikačních programů na změnách ve struktuře databáze a použitých přístupových metodách. V dalších letech se teorie relačního modelu dále rozvíjela a začala implementace prvních relačních systémů. [4]

2.3.2 Struktura relačních databází

V souvislosti s relačními databázemi se nejčastěji setkáváme s konstatováním, že data v relační databázi jsou strukturována do *tabulek*. Pojem tabulka ale nevyhovuje matematickému vyjadřování. V teorii relačního modelu dat se používá pro označení takové tabulky pojem *relace*. Příklad relace bychom mohli znázornit následující tabulkou.

LOGIN	JMÉNO	PŘÍJMENÍ	PRACOVIŠTĚ
karas00	Tomáš	Karásek	ekonomický úsek
musil01	Jan	Musil	právní úsek
vrana00	Břetislav	Vrána	právní úsek

Tabulku můžeme chápat jako tvořenou záhlavím, které se označuje jako *schéma relace*, a tělem. Matematickému pojmu relace potom odpovídá tělo tabulky. Je zřejmé, že počet řádků tabulky i jejich obsah se obecně v čase mění.

Prvky domén mohou nabývat libovolné hodnoty a to i z hlediska složitosti. Relační model dat ale klade v tomto smyslu na hodnoty domén poměrně přísné omezení – domény mohou obsahovat pouze tzv. *atomické*, označované jako *skalární*, hodnoty. Znamená to, že hodnota musí představovat z hlediska svého významu nedělitelný celek.

Relace (tabulka) na doménách obsahujících pouze skalární hodnoty se, pokud chceme tuto skutečnost zdůraznit, označuje jako *normalizovaná*. Říkáme také, že taková relace je v 1. normální formě (1NF). Naopak relace na složených či vícehodnotových formách (která ve skutečnosti není relací ve smyslu relačního modelu dat) se označuje jako *nenormalizovaná* a proces převodu nenormalizovaných relací na normalizované, splňující vlastnosti dobrého návrhu, se označuje jako *normalizace*. Důvodem pro přísný požadavek atomických hodnot v tabulce je zejména fakt, že operace s normalizovanou tabulkou jsou podstatně jednodušší než s tabulkou nenormalizovanou.

Pojem normálních forem se používá ve spojitosti s dobře navrženými tabulkami. Správně vytvořené tabulky splňují 4 základní normální formy. První normální formu jsme si popsali již výše, nyní se budeme zabývat dalšími normálními formami.

- **2NF -** Tabulka splňuje 2NF, právě když splňuje 1NF a navíc každý atribut, který není primárním klíčem, je na primárním klíči úplně závislý. To znamená, že se nesmí v řádku tabulky objevit položka, která by byla závislá jen na části primárního klíče. Z definice vyplývá, že problém 2NF se týká jenom tabulek, kde volíme za primární klíč více položek než jednu.
- **3NF -** Relační tabulky splňují třetí normální formu (3NF), jestliže splňují 2NF a žádný atribut, který není primárním klíčem, není tranzitivně závislý na žádném klíči.
- **Boyce-Coddova normální forma -** Poslední prakticky užívanou formou je tzv. Boyce-Coddova normální forma (BCNF). Tabulka splňuje BCNF, právě když pro dvě množiny

atributů A a B platí: A->B a současně B není podmnožinou A, pak množina A obsahuje primární klíč tabulky. Tato forma zjednodušuje práci s tabulkami. Ve většině případů, pokud dobře postupujeme při tvorbě tabulek, aby splňovaly postupně 1NF, 2NF a 3NF, forma BCNF je splněna. [4,6]

2.3.3 Integritní omezení v relační databázi

Protože typicky existují souvislosti mezi informacemi reálného světa uloženými v tabulkách, musíme mít možnost i tyto souvislosti v nějaké podobě do databáze uložit. U předrelačních systémů se tyto souvislosti uchovávaly v podobě ukazatelů mezi uloženými záznamy. Nevýhodou tohoto způsobu byla zpravidla závislost aplikací na cestách k datům definovaných těmito ukazateli.

V relačním modelu dat se vazba mezi dvěma řádky dvou různých tabulek řeší jako logická, vytvořená na základě rovnosti hodnot v určitých sloupcích odkazovaného a odkazujícího řádku. Aby bylo takové vazby možné vytvářet, je potřeba vyřešit dva problémy. Jednak musí existovat možnost jednoznačné identifikace odkazovaného řádku, jednak musí existovat v odkazující se tabulce sloupec, jehož hodnoty budou vazbu vytvářet. Relační model řeší tyto dva problémy pomocí tzv. *klíčů.* Přesněji pro identifikaci se používají tzv. *primární klíče* a pro odkazy tzv. *cizí klíče*. Budou to sloupce, pro jejichž hodnoty platí určitá omezení, která relační model dat definuje.

Požadavky kladená na data uložená v databázi se nazývají integritní omezení. Tato omezení musí být splněna, mají-li být data v databázi správná. Integritní omezení můžeme rozdělit do dvou skupin:

- obecná
- specifická

V prvém případě jde o integritní omezení, která musí platit v databázi relačního typu bez ohledu na konkrétní aplikační zaměření. Právě taková integritní omezení definuje relační model pro sloupce, které budou plnit roli primárních a cizích klíčů. Jde tedy o omezení, která musí takové sloupce splňovat bez ohledu na to, zda jde o relační databázi informačního systému fakulty nebo například modulu plánování výroby.

Na druhé straně každá konkrétní aplikační oblast má zpravidla svá specifická omezení na data, která jsou v databázi uložena. [4]

2.3.4 Relační algebra

Pro práci s tabulkami v relační databázi je nutné definovat alespoň základní aparát, který nám umožní zpracovávat data z tabulek. K těmto činnostem slouží prostředek nazývaný relační algebra. Relační algebra je nejzákladnějším prostředkem pro práci s tabulkami. Relační algebrou rozumíme dvojici $RA = (R, O)$, kde nosičem R je množina relací a O je množina operací. Základní operace můžeme rozdělit do dvou skupin:

- tradiční množinové operace (sjednocení, průnik, rozdíl, součin)
- speciální relační operace (projekce, selekce, spojení)

Množinové operace jsou definovány následovně:

- Sjednocením relací **R1** = *(R, R1*)* a **R2** = *(R, R2*)* se schématem *R* je relace **R1** *union* **R2** = *(R, R1* ∪ R2*).*
- Analogicky pro *průnik* (**R1** *intersect* **R2**) a *rozdíl* (**R1** *minus* **R2**).
- Kartézským součinem relací **R1** = *(R, R1*)* a **R2** = *(R, R2*)* je relace **R1** times **R2** = *((R1,R2), R1* x R2*).* [4,5]

Následují příklady, pro něž uvažujme následující tabulky T1, T2

T1

A	B	C
0	a	d
1	a	e
2	b	f

T2

A	B	C
2	c	d
0	a	d
0	a	e

- tabulka získaná sjednocením T1 s T2

T1 union T2

A	B	C
0	a	d
1	a	e
2	b	f
2	c	d
0	a	e

- tabulka získaná průnikem T1 s T2

T1 intersect T2

A	B	C
0	a	d

- tabulka získaná rozdílem T1 a T2

T1 minus T2

A	B	C
1	a	e
2	b	f

- výsledek operace kartézského součinu (pro rozlišení sloupců tabulek mají ve výsledku sloupce koncovku T1, resp. T2)

T1 times T2

AT1	BT1	CT1	AT2	BT2	CT2
0	a	d	2	c	d
0	a	d	0	a	d
0	a	d	0	a	e
1	a	e	2	c	d
...	...	...	...	...	...

Speciální relační operace jsou definovány následovně.

2.3.4.1 Projekce

Projekce relace **R** = (*R*, *R**) na atributy *X*, *Y*, ..., *Z* je relace **R**[*X*, *Y*, ..., *Z*] se schématem (*X*, *Y*, ..., *Z*) a tělem zahrnujícím všechny n-tice *t* = (*x*, *y*, ..., *z*) takové, že v *R** existuje n-tice *t´* s hodnotou atributu *X* rovnou *x*, *Y* rovnou *y*, ... *Z* rovnou *z*.

Výsledkem projekce tedy bude tabulka, která bude obsahovat jenom některé ze sloupců původní tabulky.

- tabulka vzniklá projekcí T2[A,B]

T2 [A,B]

A	B
2	c
0	a

2.3.4.2 Selekce

Nechť Θ je operátor porovnání dvou hodnot (<, >, <>, =, atd.). Θ selekce relace **R** = (*R*, *R**) na atributech *X* a *Y* je relace **R** *where* $X\Theta Y$, která má stejné schéma jako relace **R** a obsahuje všechny n-tice $t \in R^*$, pro které platí $x \Theta y$, kde *x* je hodnota atributu *X* a *y* hodnota atributu *Y* v n-tici *t*. Na místě atributu *X*, resp. *Y* může být konstanta. Pak jde o Θ selekci na atributu *X*, resp. *Y*.

Výsledkem operace selekce bude narozdíl od projekce tabulka, která bude zahrnovat všechny sloupce, ale obecně jen některé řádky. Budou to ty řádky, které vyhovují dané podmínce.

- tabulka vzniklá selekcí T3 *where* A<B

T3

A	B	C
0	1	d
1	1	e
2	1	f

T3 where A < B

A	B	C
0	1	d

2.3.4.3 Přirozené spojení

Poslední ze speciálních relačních operací je spojení, přesněji přirozené spojení. Umožňuje nám spojovat řádky dvou tabulek na základě stejné hodnoty ve stejně pojmenovaných sloupcích. Formálně bychom ji mohli definovat následovně.

Nechť **R1** = (*R1, R1**) je relace se schématem *R1(X1, X2, ..., Xm, Y1, Y2, ..., Yn)* a **R2** relace se schématem *R2(Y1, Y2, ..., Yn, Z1, Z2, ..., Zk).* Uvažujme složené atributy *X = (X1, X2, ..., Xm), Y = (Y1, Y2, ..., Yn), Z = (Z1, Z2, ..., Zm).*

Potom přirozené spojení relací **R1** a **R2** je relace **R1** *join* **R2** se schématem (*X, Y, Z*) a tělem zahrnujícím všechny n-tice *t* = (*x, y, z*) takové, že v *R1** existuje n-tice *t´* s hodnotou *x* atributu *X* a hodnotou *y* atributu *Y* a v *R2** existuje n-tice *t´´* s hodnotou *z* atributu *Z.*

- tabulka vzniklá přirozeným spojením T4 a T5

T4

A	B	C
0	a	d
1	a	e
2	b	f

T5

C	D	E
e	1	0
d	1	1
d	0	1

T4 join T5

A	B	C	D	E
0	a	d	1	1
0	a	d	0	1
1	a	e	1	0

Spojení probíhá na základě rovnosti hodnot ve sloupci C. Prvý řádek tabulky T4 se proto spojí s druhým a třetím řádkem tabulky T5 a vzniknou tak prvé dva řádky výsledné tabulky. Třetí řádek výsledku vznikne spojením druhého řádku tabulky T4 a prvého řádku tabulky T5. Poslední řádek T4 se nespojí s žádným řádkem tabulky T5, a proto se jeho hodnoty ve výsledné tabulce neobjeví.

2.4 SQL

SQL je standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích. SQL je zkratka anglických slov Structured Query Language (strukturovaný dotazovací jazyk). Tímto jazykem lze kompletně manipulovat s tabulkami v databázi a v nich uloženými daty. [7,8]

2.4.1 Historie SQL

V 70. letech 20. století probíhal ve firmě IBM výzkum relačních databází. Bylo nutné vytvořit sadu příkazů pro ovládání těchto databází. Vznikl tak jazyk SEQUEL (Structured English Query Language). Cílem bylo vytvořit jazyk, ve kterém by se příkazy tvořily syntakticky co nejblíže přirozenému jazyku (angličtině).

K vývoji jazyka se přidaly další firmy. V r. 1979 uvedla na trh firma Relational Software, Inc. (dnešní Oracle Corporation) svoji relační databázovou platformu Oracle Database. IBM uvedla v roce 1981 nový systém SQL/DS a v roce 1983 systém DB2. Dalšími systémy byly např. Progress, Informix a SyBase. Ve všech těchto systémech se používala varianta jazyka SEQUEL, který byl přejmenován na SQL.

Relační databáze byly stále významnější, a bylo nutné jejich jazyk standardizovat. Americký institut ANSI původně chtěl vydat jako standard zcela nový jazyk RDL. SQL se však prosadil jako de facto standard a ANSI založil nový standard na tomto jazyku. Tento standard bývá označován jako SQL-86 podle roku, kdy byl přijat.

V dalších letech se ukázalo, že SQL-86 obsahuje některé nedostatky a naopak v něm nejsou obsaženy některé důležité prvky týkající se hlavně integrity databáze. V roce 1992 byl proto přijat nový standard SQL-92 (někdy se uvádí jen SQL2). Zatím nejnovějším standardem je SQL3 (SQL-99), který reaguje na potřeby nejmodernějších databází s objektovými prvky.

Standardy podporuje prakticky každá relační databáze, ale obvykle nejsou implementovány vždy všechny požadavky normy. A naopak, každá z nich obsahuje prvky a konstrukce, které nejsou ve standardech obsaženy. Přenositelnost SQL dotazů mezi jednotlivými databázemi je proto omezená. [7]

2.4.2 Popis jazyka

SQL příkazy se dělí do čtyř základních skupin:

- příkazy pro manipulaci s daty
- příkazy pro definici dat
- příkazy pro řízení dat
- ostatní příkazy

Následuje stručný popis každé skupiny.

2.4.2.1 Příkazy pro manipulaci s daty

Jsou to příkazy pro získání dat z databáze a pro jejich úpravy. Označují se zkráceně DML – Data Manipulation Language („jazyk pro manipulaci s daty“).

Příkazy:

- SELECT – vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat.
- INSERT – vkládá do databáze nová data.
- UPDATE – mění data v databázi (editace).
- DELETE – odstraňuje data (záznamy) z databáze.
- EXPLAIN PLAN FOR – speciální příkaz, který zobrazuje postup zpracování SQL příkazu. Pomáhá uživateli optimalizovat příkazy tak, aby byly rychlejší.
- SHOW - méně častý příkaz, umožňující zobrazit databáze, tabulky nebo jejich definice[7]

2.4.2.2 Příkazy pro definici dat

Těmito příkazy se vytvářejí struktury databáze – tabulky, indexy, pohledy a další objekty. Vytvořené struktury lze také upravovat, doplňovat a mazat. Tato skupina příkazů se nazývá zkráceně DDL – Data Definition Language („jazyk pro definici dat“).

Příkazy:

- CREATE – vytváření nových objektů.
- ALTER – změny existujících objektů.
- DROP – odstraňování objektů. [7]

2.4.2.3 Příkazy pro řízení dat

Do této skupiny patří příkazy pro nastavování přístupových práv a řízení transakcí. Označují se jako DCL – Data Control Language („jazyk pro ovládání dat“), někdy také TCC – Transaction Control Commands („jazyk pro ovládání transakcí“).

Příkazy:

- GRANT – příkaz pro přidělení oprávnění uživateli k určitým objektům.
- REVOKE – příkaz pro odnětí práv uživateli.
- BEGIN – zahájení transakce.
- COMMIT – potvrzení transakce.
- ROLLBACK – zrušení transakce, návrat do původního stavu. [7]

2.4.2.4 Ostatní příkazy

Do této skupiny patří příkazy pro správu databáze. Pomocí nich lze přidávat uživatele, nastavovat systémové parametry (kódování znaků, způsob řazení, formáty data a času apod.). Tato skupina není standardizována a konkrétní syntaxe příkazů je závislá na databázovém systému. V některých dialektech jazyka SQL jsou přidány i příkazy pro kontrolu běhu, takže lze tyto dialekty zařadit i mezi programovací jazyky. [7]

3 Analýza portálu VUT

Tato kapitola popisuje analýzu portálu VUT. Nejprve jsou uvedeny základní požadavky na tento portál. V další části je pak nastíněno jak je portál realizován. Některé technologie (HTML, PHP, JavaScript, Oracle), na kterých je portál založen, nejsou v této kapitole podrobně rozebrány. Tomu se věnuji až v kapitole, která se týká implementace.

3.1 Požadavky na portál

Na univerzitní portál VUT je kladeno několik základních požadavků. Portál by měl současně sloužit jako prezentační část webu VUT pro cizí návštěvníky i jako primární zdroj informací pro jeho přihlášené klienty. Konkrétně se jedná o zaměstnance VUT, studenty, pedagogy a další.

Důležitým požadavkem je schopnost spravovat co největší počet školských agent. Pro studenty je například důležité vyhledat a nastavit si svůj osobní studijní rozvrh, stáhnout si potřebné informace o zapsaných předmětech včetně materiálů, předměty si zapisovat, dále se v nich přihlašovat a odevzdávat projekty a samostatné práce, zapisovat termíny zkoušek a v neposlední řadě získávat informace o ohodnocení daného předmětu a jeho částí centralizovaně na jednom místě. Teprve pak se portál stane neocenitelným pomocníkem studenta při jeho studiu. V současné době je trendem zavádění tzv. elektronických indexů, kdy veškeré hodnocení a kontrola studia probíhá pomocí informačního systému a fyzické indexy se tak stávají zbytečností. Tento přístup také přináší lepší možnost kontroly studia studenta.

Na základě těchto aktivit musí pochopitelně také existovat nějaké rozhraní pro pedagogy, aby mohli studijní agendu ovládat ze svého úhlu ohledu (např. vypisování termínů zkoušky, zadávání hodnocení do systému). Existuje mnoho agend, které je nutné spravovat, namátkou například publikace, projekty, technologie, výrobky, služby, konference, semináře, poskytování informací o platbách univerzitní identifikační kartou, nebo třeba pro zaměstnance výpisy a vyúčtování telefonních hovorů.

Portál by měl také spravovat osobní informace o všech uživatelích, kteří jsou schopni se přihlásit, a vhodně je prezentovat.

Dalším požadavkem je obsažení publikačního systému, který by byl využitelný podle práv pro všechny uživatele. S ním souvisí i přítomnost jiných prvků pro získávání informací od uživatelů jako jsou diskusní fóra a ankety. Single Sign On musí umět zajistit bezpečné přihlášení na základě tzv. VUT loginu a toto přihlášení by mělo být jednotné při pohybu po všech možných částech portálu bez nutnosti opakovaného ověřování identity, které zpomaluje práci a působí negativně na uživatele.

Musí být dostupná plná přenositelnost na jiné stroje, aniž by se muselo příliš zasahovat do zdrojových kódů. [9]

3.2 Používaný software a knihovny

3.2.1 Databáze Oracle

Primárním zdrojem dat je několik instancí databáze Oracle. V současnosti bychom těžko našli výkonnější databázový stroj. Jedna z instancí je používána jako ostrá verze, další jsou pro testovací a vývojové užití. Databáze běží na samostatném serveru pro tento účel vyhrazený. Kromě klasického schématu tabulek, klasických a materializovaných pohledů se ještě používají procedury a funkce, což je doména Oraclu. Tyto procedury a funkce, vytvořeny jazykem plsql, jsou vynikajícím prostředkem k dolování informací z databáze, jelikož hlavní dotazy jsou centralizované v databázi a klientské aplikace volají pouze rozhraní procedury. Bohužel se však s jejich nasazením otálelo, takže se často setkáme s sql dotazy ve zdrojových kódech. [9]

3.2.2 Subversion

Subversion je modul na serveru pro správu jednotlivých verzí zdrojových souborů. Jedná se o systém, který udržuje kontrolu nad skupinou souborů, podchytává veškeré změny, které byly na souborech provedeny a v případě potřeby umožňuje kdykoli návrat k jakékoli dřívější verzi. Zároveň umožňuje práci více vývojářů na jedné aplikaci, či zdrojovém kódu. [9]

3.2.3 ADODB

Jelikož přístup z jazyka PHP do databází není standardizován a je implementován pro každou databázi rozdílně, bylo vhodné pro účely případných změn a zjednodušení přístupu ke zdrojům dat použít PHP knihovny ADOdb. ADOdb (Database Abstraction Library for PHP - Python) tvoří abstraktní vrstvu mezi databázovými stroji a uživatelskými skripty, které pracují pomocí SQL jazyka se zdroji databází. Hlavní efekt přináší smazání rozdílů volání API funkcí pro jednotlivé databázové stroje.

V prvních verzích portálového řešené na VUT se tato vrstva nepoužívala a dotazy na centrální databázi se volaly přímo pomocí vestavěných funkcí v PHP a ovladače OCI8. Poté se přešlo na podpůrnou knihovnu řešící konektivitu do databází ADODB. Neocenitelnou výhodou této knihovny je absolutní přenositelnost. Mezi další výhody patří přehlednost a hlavně možnost kdykoli změnit typ databáze přepsáním jediného parametru při připojováni databázi. Jedinou podmínkou je nutnost stejné datové struktury. Tímto se stává ADODB důležitým prvkem při vývoji, jelikož umožňuje vývoj aplikace mimo hlavní prostor portálu, klidně i lokálně na databázi mysql u vývojáře. Pak stačí jen zkopírovat odladěné zdrojové kódy a přepsat parametr ovladače na jiný typ databáze. [9]

3.2.4 JpGraph

Knihovna JpGraph je výborný nástroj pro zobrazování grafů. Dokáže, pomocí knihovny GD2, která je součástí kompilace apache, vykreslit velké množství různorodých grafů, od základních jedno až x-řadových sloupcových, křivkových, koláčových až po polární, tří-dimenzionální, obrázkové či mapové. Do zdrojového kódu PHP je vždy nutno vložit příslušnou část knihovny, nadefinovat vstupní pole dat, popisků a o zbytek se postará vykreslující funkce knihovny. Výsledné grafy jsou ve formátu png obrázků, knihovna je freeware.[9]

3.2.5 Smarty

Nadstavba nad PHP modul Smarty řeší základní problém většiny implementací a to promíchání zdrojových kódů s generováním výstupu, obvykle HTML. Smarty vytváří generovaný kód na základě PHP kódu definující proměnné a grafické šablony v souboru tpl, která je napsaná pomocí speciálních smarty direktiv. Ty umožňují snadný výpis polí, formulářových prvků, dat a podobně.

Svým pojetím je smarty ideálním prostředkem pro spolupráci grafiků kodérů HTML a programátorů dolujících informace z různých zdrojů. Programátor a kodér se domluví na názvech polí a proměnných s daty, které se zobrazí. Pokud dodrží toto rozhraní, nemusí navzájem nic vědět o práci toho druhého, needitují stejné soubory a přehlednost v kódech se tak výrazně zvýší.

Smarty bylo testováno na portálu několik měsíců, ale bohužel se neosvědčilo z důvodu rychlosti vygenerování a zobrazení stránky v prohlížeči, pokud stránka obsahovala více dat a byla složitější na vygenerování. Obdobný kód generovaný čistě pomocí PHP byl hotov rychleji než pomocí smarty. Navíc nešlo řešit a nastavit některé základní parametry značek jazyka HTML. Tento přístup je tedy pro portál nevyužitelný, jelikož by značně trpěla rychlost systému, což je jedním z hlavních požadavků. [9]

3.2.6 PDFlib

PDFlib je volně šiřitelná knihovna pro generování pdf dokumentů přímo z PHP. Obsahuje několik formátovacích metod, podle kterých se data zobrazí ve výsledně formě. PDFlib je napsána v jazyce PHP a jejích rozhraní se vkládá do php kódu. Je využívána primárně ke generování přehledů pro tisk k mnoha aplikacím. [9]

3.3 Systémové prostředky, hardware

3.3.1 Databázový server

Celý systém portálu se skládá z několika serverů, které se starají o základní funkce a navzájkem spolu spolupracují. Servery jsou mezi sebou propojeny vnitřním síťovým okruhem s rychlostí 1Gbit. Výčet

serverů s jejich funkcemi: Hlavní databázový server OGRE a testovací server OGRENESS provozují centrální databázi na platformě Oracle, která obsahuje celou agendu vysoké školy včetně schématu pro provoz portálu. Jedná se o 4-procesorový stroj na platfomrě Intel XEON s 16GB operační pamětí a poněkud slabší verze pro server OGRENESS. Operačním systémem je distribuce linuxu RedHat. [9]

3.3.2 Aplikační cluster serverů

Vzhledem k důležitosti provozu webových stránek a informačního systému vysoké školy je nutno zajistit nepřetržitou dostupnost a z ní vyplývající vysokou odolnost služby proti výpadku a omezit tak možnost selhání sytému. Řešením je použití clusteru serverů. Cluster funguje obecně tak, že existuje více webových serverů, které jsou však schovány před uživatelem a navenek se cluster tváří jako jeden server. Uvnitř clusteru se efektivně rozdělují požadavky na zpracování mezi jednotlivé aplikační servery, například podle vytížení.

Pro běh informačních systémů VUT je použito řešení RedHat Cluster System, kde podpora pro provoz clusteru je zakomponována přímo v jádře distribuce RedHat Linux. Tato služba se jmenuje IPVSADM a zajišťuje přerozdělování a přesměrování všech TCP připojení a UPD datagramů na fyzické servery podle desíti integrovaných algoritmů, které jsou distribuovány s Linux Virtual Server (LVS). Přesměrování požadavků zajišťuje server piranha, na kterém IPVSADM běží. [9]

3.3.3 Podpůrné servery

PC-ldap je autentizační server, starající se o navazování, ověřování a udržování bezpečného přihlášení.

Dále je ještě do sítě zapojeno několik zálohovacích serverů, které v pravidelných časových intervalech vytvářejí kopie disků všech serverů na disková pole a na magnetické pásky. Pro samotný chod systému však nejsou třeba. V případě havárie či selhání systému jsou však neocenitelné. [9]

3.4 Softwarové požadavky

Pro správnou funkci portálu musí být na serveru nainstalován operační systém Linux, webový server apache s modulem csacek, podporou knihovny GD2.0, ovladačem OCI8 pro připojení databáze, překladačem PHP. Dále je nezbytný server s databází Oracle, U klientů je vyžadován internetový prohlížeč s podporou normy HTML 4.01 transitional a CSS 2.1 se zapnutou podporou cookies a javasciptu. V případě nižší verze nemusí být funkčnost systému zaručena. [9]

3.5 Struktura portálu

Struktura portálu je odvozena od jeho hlavních součástí a požadovaných funkcí. Obsahuje soubory knihoven s obecnými funkcemi, které jsou využívány v celém projektu, např. funkce pro práci s databázovým systémem Oracle, či pro vložení titulu stránky.

Hlavním jádrem je soubor index.php přes který se parametrem určuje stránka, která se má zobrazit, nebo aplikace, která se má spustit. Tento soubor ověřuje přítomnost stránky či aplikace, dále vkládá podle kontextu menu vlevo, či informační panel vpravo. Také načítá příslušný kaskádový styl stránky. Neméně důležité jsou hlavičkový a patičkový soubor. Tyto soubory obsahují obecný html kód definující základní vlastnosti rozdělení portálu. Jsou kompletně napsány pro zformátování pomocí css souboru a to včetně obrázků.

Souborů s kaskádovými styl je zde více. Jeden je základní a pak každý další odpovídá jednomu webu, který běží na engine portálu, ale není přímo webem VUT. Většinou se jedná o webové stránky součástí (např. Centra výpočetních a informačních služeb, Útvaru pro transfer technologií, Ústřední knihovny, kompetenčního centra VUT a intraportálu).

Mezi další důležité části patří soubory pro funkci publikačního systému, na kterém celý portál stojí, a adresář obsahující jednotlivé aplikace, které lze pod portálem spustit ve formě stránky nebo portletu. [9]

3.6 Autentizace

Autentizace je prováděna pomocí protokolu LDAP a to vůči autentizačnímu serveru. Princip LDAPu je založen na stromové struktuře tzv. organizačních jednotek, organizačních podjednotek, pracovních skupin a uživatelů. Vrcholem stromu je vždy organizační jednotka, která může obsahovat další organizační jednotky a pracovní skupiny. Pracovní skupiny jsou jen pod organizačními jednotkami a mohou obsahovat jen výčet uživatelů. Limity počtů organizačních jednotek, pracovních skupin a uživatelů nejsou omezeny. Ke každé organizační a pracovní skupině existuje speciální jednotka organizátorů, kteří mohou vytvářet a mazat další skupiny, jednotky, a uživatele kontextově v další části struktury, pod touto jejich jednotkou či skupinou. Prakticky může být jeden uživatel součástí několika skupin, takže tímto způsobem lze prakticky nastavovat práva.

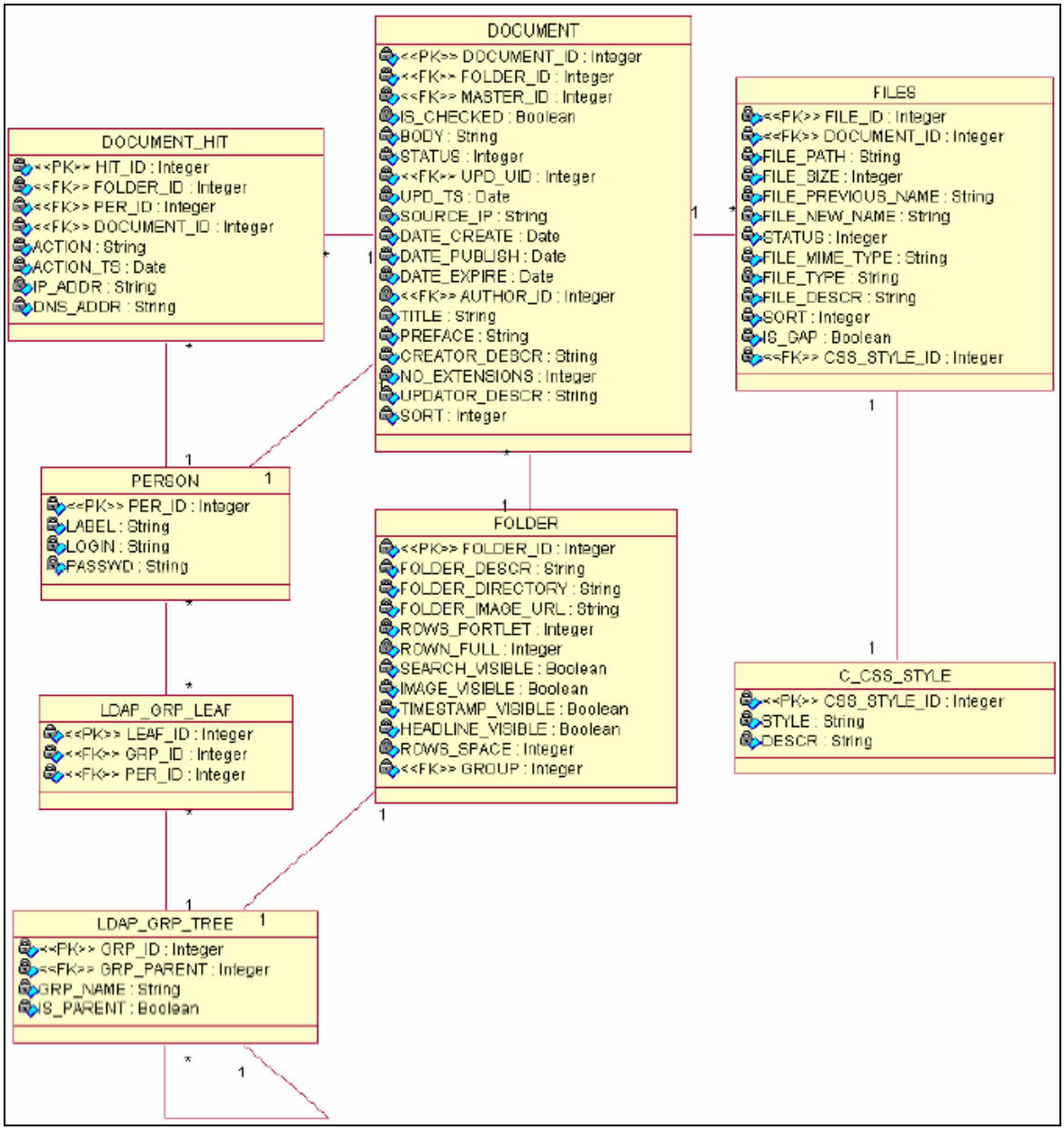
Pro účely portálu existuje jedna úroveň organizačních jednotek, které obsahují maximálně čtyři pracovní skupiny (1, 10, 100, 1000) ve smyslu práv prohlížet, přidávat, mazat své, publikovat. Tyto práva jsou prioritně spojena s funkcí publikačního systému a jsou kaskádově spojena, Tzn. má-li někdo právo publikovat, má současně i všechna práva nižší priority. Podle tohoto seznamu lze vygenerovat pole práv a to pak ověřovat s konkrétní aplikací.

Pro editaci LDAP skupin, jednotek a uživatelů existuje webové rozhraní s intuitivním ovládáním. Dále je pak k dispozici i konzole ldap napsaná v jazyce Java.

Informace o uživatelích se do databáze replikují z centrálního systému v každou celou hodinu.[9]

3.7 Datový sklad

Pro funkci portálu se používá několik tabulek převážně publikačního systému. Zato aplikace využívají různé části databáze, podle typu agendy, které spravují. ER model publikačního systému je zobrazen na obrázku 3.1. Znázorňuje spojení základních tabulek dokumentu, složek, které je sdružují a vazby na souborové přílohy a tabulku person se základními údaji o lidech na VUT. [9]



Obrázek 3.1: ER schéma jádra publikačního portálu VUT [9]

3.8 Publikační systém

Hlavním prvkem je obecný dokument, který krom několika atributů, nadpisu, těla a úvodu dokumentu může sdružovat různé přílohy ve formátu mime. Je povoleno několik typů nejpoužívanějších dokumentových formátů a některé formáty multimediální. Dokumenty jsou sdružovány do složek, které lze vypsat několik způsoby, včetně podoby portletu s nejaktuálnějšími dokumenty na prvních místech. Pořadí příloh a dokumentů ve složce lze libovolně měnit pomocí webového rozhraní.

Publikační systém využívá systému práv v ldapu pro přístup do jednotlivých složek a rozhoduje tak, jaké operace uživateli povolí nad danou složkou vykonat.

Tělo každého dokumentu může být libovolný html kód, který lze vložit přímo, nebo prostřednictvím javascriptového editoru na webu nebo zkopírovat z externího editoru. [9]

3.9 Aplikace

Aplikace na portálu jsou umístěny v samostatném adresáři app. Většinou je přítomna jedna knihovna ke každé aplikaci, pomocí které se daná aplikace ovládá. Odkaz na tuto knihovnu je přidán do výčtu povolených souborů v index.html.[9]

4 Datagridy

Datagrid je grafické uživatelské rozhraní, které uživateli umožňuje pracovat s daty, která jsou uloženy v databázi. Tato komponenta vždy načte data z databáze a následně je zobrazí uživateli v podobě přehledné tabulky. Uživatel s těmito daty pak může následně pracovat. Největší výhodou této komponenty je bezesporu to, že uživatel, který s ní pracuje, je oproštěn od databázového schématu. Nemusí s daty v databázi pracovat prostřednictvím SQL dotazů, to za něho zajistí právě datagrid. S touto komponentou tak dokáže pracovat i naprostý laik, který nemá ponětí o teorii databázových systémů.

Dnes již existuje řada komponent, které dokáží pracovat s tabulkovými daty. Tyto komponenty pracují s různými databázemi (Oracle, MySQL). Filosofie však zůstává stále stejná, poskytnout uživateli možnost jak jednoduše pracovat s daty v databázi.

Většina těchto komponent je zobrazována pouze pomocí webového prohlížeče, což přináší další výhody. Nový uživatel nemusí instalovat nový program, dále odpadají problémy s obsluhou programu, protože řada uživatelů práci s webovým prohlížečem lehce zvládá. Jelikož jsou data zobrazovány v tabulce, práce s daty je vždy intuitivní. Nejnovější datagridy jsou napsány v různých programovacích jazycích. Asi nejvíce rozšířenou se stala implementace v jazyce PHP. Na straně klienta některé datagridy používají JavaScript pro kontrolu odesílaných formulářů, pro řazení či filtry. Nejnovějším trendem je použití technologie AJAX. Pomocí AJAXu lze při změně dat v tabulce tyto data odesílat na server na pozadí. Odpadají tak prodlevy způsobené opětovným načtením celé stránky. Načte se vždy jen ta část, která je modifikována. Tuto skutečnost jistě uživatel ocení, protože nemusí čekat, až se znovu načte celá zobrazovaná stránka a může plynule pokračovat ve své práci.

Existují i komponenty, které mají své vlastní prostředí (neovládají se pomocí webového prohlížeče). Protože jsou data i zde vždy zobrazována kvůli přehlednosti v tabulce, není ani práce s těmito komponentami uživatelsky náročná.

Současné datagridy kromě přehledného zobrazení dat z databáze umožňují uživateli řadu možností jak s těmito daty dále pracovat.

Mezi základní vlastnosti každého datagridu patří stránkování výsledné zobrazované tabulky. Pokud je počet zobrazovaných řádků tabulky příliš velký, bylo by nepřehledné zobrazovat všechny řádky naráz na jedné stránce. Zobrazí se tedy pouze část tabulky a zároveň jsou zobrazovány informace o tom, které řádky jsou právě zobrazovány, a odkazy na další části tabulky. Pomocí těchto odkazů lze mezi částmi tabulky libovolně procházet. Většinou také datagrid poskytuje možnost nastavení kolik řádků tabulky je zobrazováno na jedné stránce. Potom si uživatel může výpis řádků přizpůsobit svým potřebám.

Dále datagrity poskytují filtry, pomocí nichž lze omezit zobrazovaná data. Existuje hodně typů těchto filtrů. Některé datagrity například umožňují zobrazit pouze řádky tabulky, které obsahují daný

podřetězec zadaný filtrem. Jiné umožňují přiřadit filtr k daném sloupci. Zobrazují se pak jen ty řádky u kterých daný sloupec vyhovuje přiřazenému filtru. Sloupce obsahující číselný typ lze omezit porovnávací podmínkou (>, >=, =, <>, <=, <). U sloupců obsahující řetězce lze například nastavit, aby se zobrazovali pouze ty sloupce, které začínají, obsahují či končí na daný výraz.

Datagrity také poskytují funkce pro řazení dat. Uživatel si v případě potřeby může data seřadit podle hodnot některého ze sloupců tabulky. Většinou lze řadit v nezávislosti na typu dat v sloupci. Je jedno jestli hodnoty v sloupcích jsou řetězce, data, či čísla. Většina datagridů bere typ sloupců v potaz a data seřadí do správného pořadí.

Kromě změny zobrazení tabulkových dat, datagridy poskytují i možnost změny dat v databázi. Data lze modifikovat, přidávat do tabulky nové řádky, nebo řádky mazat. Vyspělé datagrity současně kontrolují integritní omezení, které jsou nad danou databází definovány. Například při vkládání nového řádku je testováno zda vkládaná hodnota koresponduje s příslušným sloupcem tabulky.

V další časti uvedu nejznámější komponenty pracující s tabulkovými daty. Protože tyto komponenty poskytují uživateli, až na hrstku odlišností, zhruba stejné funkce, které jsou již výše popsány, uvedu u každé komponenty jen její základní charakteristiku doplněnou obrázkem, na kterém je daná komponenta zobrazena.

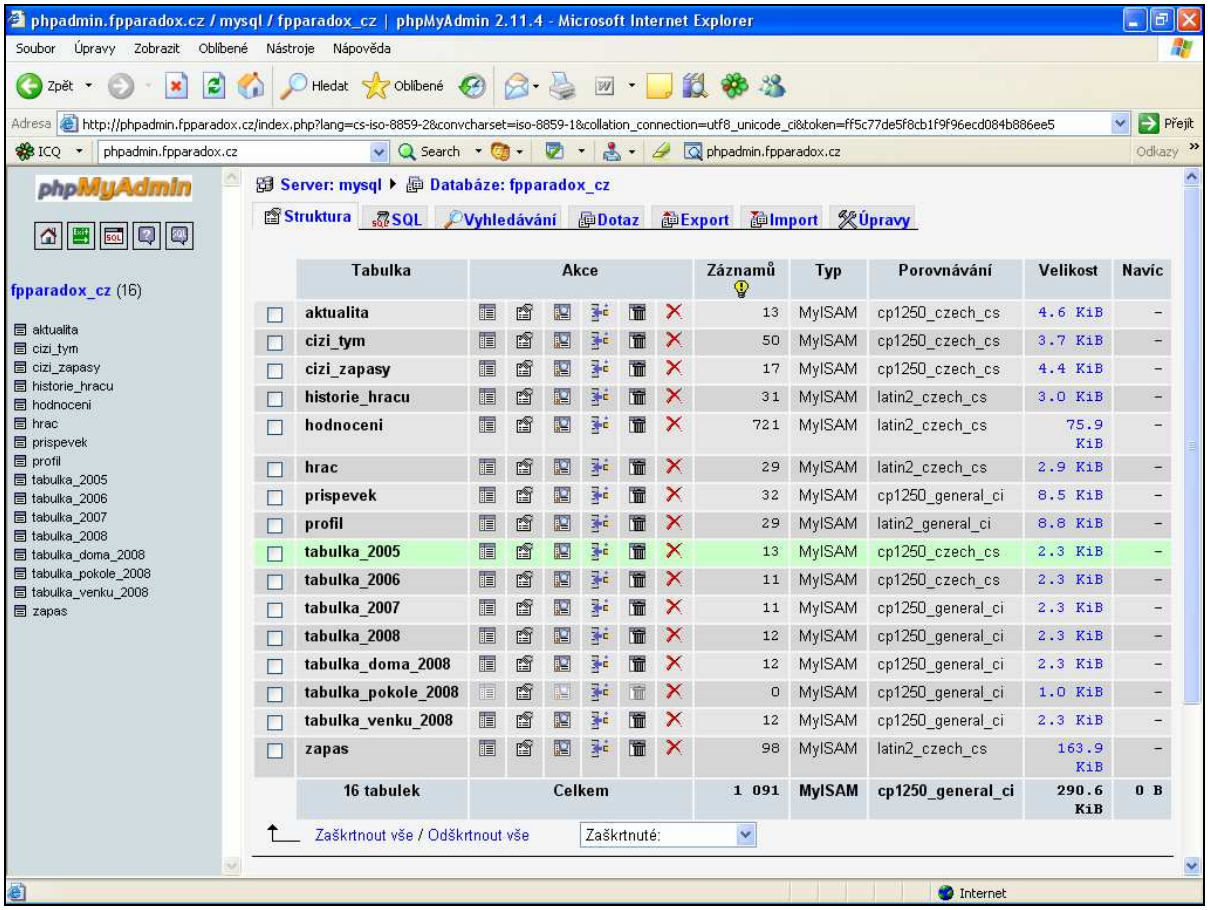
4.1 phpMyAdmin

phpMyAdmin je nástroj napsaný v jazyce PHP umožňující jednoduchou správu obsahu databáze MySQL prostřednictvím webového rozhraní. V současné době umožňuje vytvářet/rušit databáze, vytvářet/upravovat/rušit tabulky, provádět SQL příkazy a spravovat klíče. Jedná se o jeden z nejpopulárnějších nástrojů pro správu databáze. Je k dispozici v 52 jazycích.

Obdobnou funkcionalitu, ale pro databázi PostgreSQL, poskytuje nástroj phpPgAdmin, který původně vznikl jako fork phpMyAdminu, ale nyní je to úplně jiný produkt.

Existuje též phpMSAdmin, který je určen pro Microsoft SQL Server. Přestože má podobný vzhled, s phpMyAdminem nemá nic společného a byl napsán kompletně od základů.

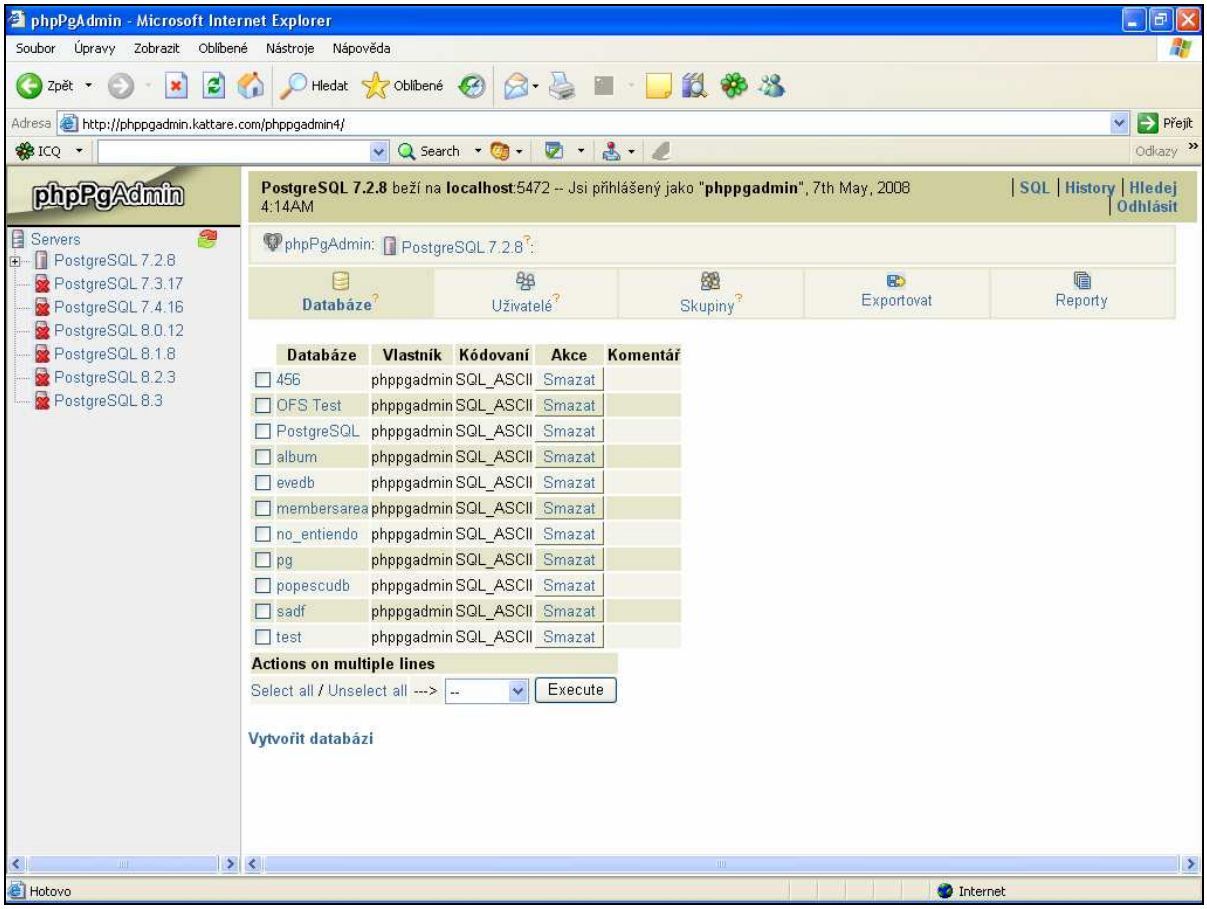
Kompaktní alternativou PhpMyAdmina je phpMinAdmin coby jeden cca 150 kB skript. [10]



Obrázek 4.1: Vzhled nástroje phpMyAdmin

4.2 phpPgAdmin

phpPgAdmin je webová aplikace napsaná v jazyce PHP, která slouží ke správě databázového serveru PostgreSQL. Disponuje funkcemi pro správu databází, práci s tabulkami, indexy, přímým zadáváním SQL dotazů. Aplikace vznikla v roce 2002 jako fork phpMyAdminu s cílem nabídnout podobné funkce pro PostgreSQL. Nyní jsou to však dva rozdílné produkty. Jedná se o nejpoužívanější webovou aplikaci pro správu PostgreSQL. Stejně jako phpMyAdmin existuje několik jazykových mutacích. [11]

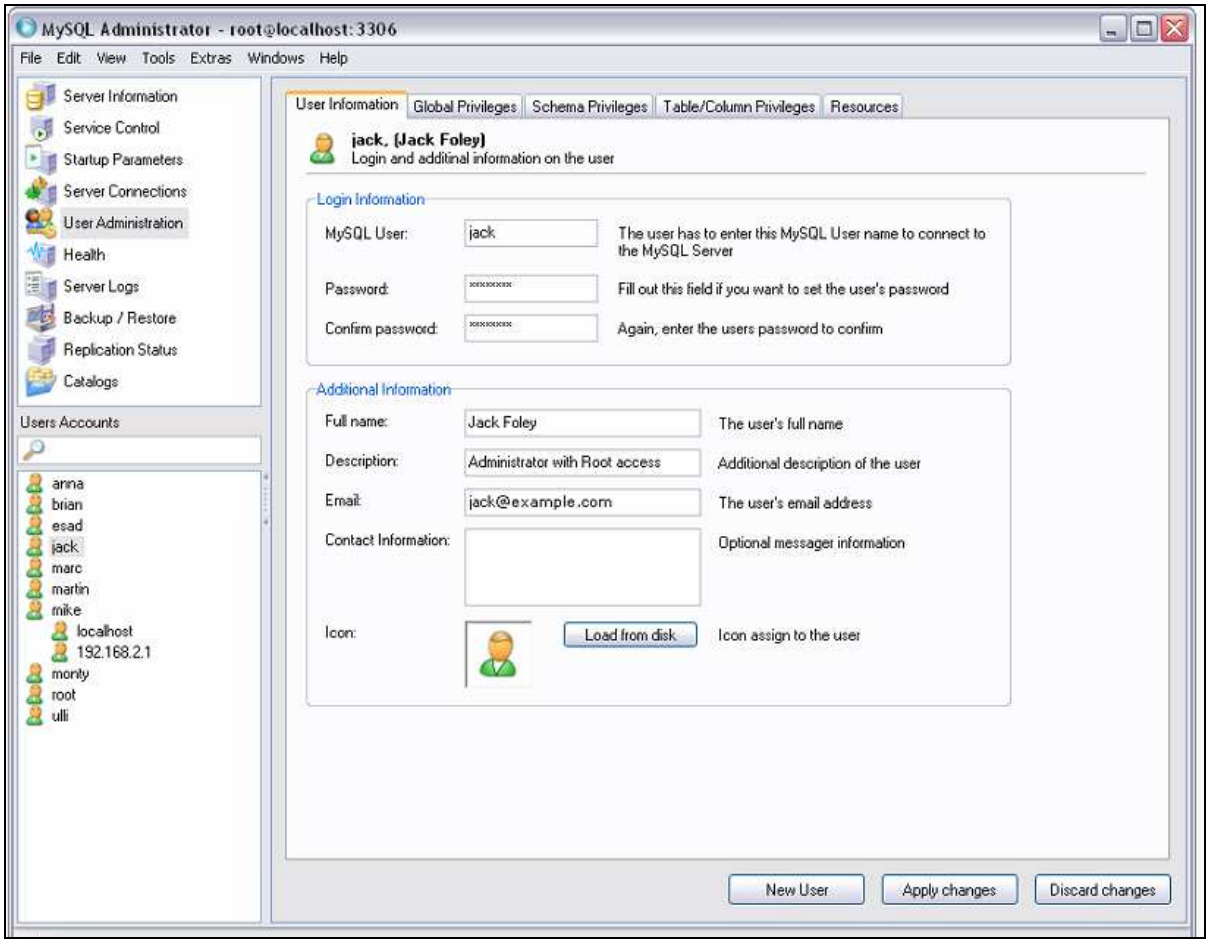


Obrázek 4.2: Vzhled nástroje phpPgAdmin

4.3 MySQL Administrator

MySQL Administrator je výkonná vizuální administrační konzola, která umožňuje snadnou administraci prostředí databáze MySQL a získat významně lepší viditelnost toho, jak databáze pracuje. MySQL Administrator umožňuje jak správu tak údržbu databáze.

Dále MySQL Administrator poskytuje vývojářům snadné vykonávání příkazů, konfiguraci serverů, správu uživatelů a dynamické sledování stavu databáze.



Obrázek 4.3: Vzhled nástroje MySQL Administrator

4.4 Další komponenty

Existují i další komponenty, které nepracují přímo s databází, ale poskytují funkce pro jednoduché modifikace dat v tabulce. Většina z těchto webových komponent edituje data pomocí JavaScriptu, který modifikuje objektový model dokumentu DOM. Dále uvedeme některé komponenty, které lze využít pro zobrazování dat databáze.

- ActiveWidgets - http://www.activewidgets.com/
- phpgrid - http://www.phpgrid.com/
- Data Grid - free grid control - http://www.codeproject.com/KB/vb-interop/Data_Grid.aspx
- Gurt JavaScript Table - http://gurtom.com/products/tables/js/free/
- XAJAX PHP Live Datagrid – http://ajaxian.com/by/topic/examples/

Některé z těchto produktů jsou komerční, jiné jsou volně stažitelné pro vlastní využití.

Na internetu je mnoho komponent, které řeší správu databáze či poskytují řadu funkčních prostředků pro práci s daty v tabulce. Tyto komponenty jsou opravdu různorodé. Pracují se spoustou typů databází a jsou implementovány v různých programovacích jazycích.

Analyzoval jsem existující komponenty na webu a zaměřil se především na funkčnosti, které poskytují při práci s tabulkovými daty. Získané poznatky jsem následně využil při návrhu a implementaci vytvářené komponenty.

5 Návrh komplementy

Tato kapitola popisuje návrh komponenty. Nejprve jsou uvedeny všechny základní požadavky kladené na vytvářenou komponentu, poté následuje rozbor možných řešení implementace jednotlivých částí komponenty s uvedením mého zvoleného řešení.

5.1 Základní požadavky

Vytvářená komponenta má uživateli umožňovat, pomocí webového prohlížeče, snadnou práci s daty uloženými v databázi Oracle. Data, se kterými následně bude uživatel pracovat, se budou specifikovat pomocí vstupního SQL dotazu. Podle tohoto dotazu je uživateli zobrazena tabulka, kterou může v případě potřeby dále editovat. Komponenta musí umět zpracovat i dotaz nad více propojenými tabulkami. Uživatel též bude moci, při zadávání vstupního dotazu, určovat, jestli zobrazenou tabulku bude možno editovat.

Nad zobrazenou tabulkou bude moci uživatel provádět různé operace. Bude možné přidávat, mazat či jinak modifikovat řádky tabulky, tabulku řadit podle různých sloupců. Dále komponenta bude umožňovat výběr zobrazovaných sloupců a možnost použití filtrů nad různými sloupci. V případě tabulek s velkým počtem řádků bude komponenta poskytovat stránkování, kdy se zobrazí jen část tabulky.

Vytvářená komponenta bude implementována v programovacím jazyce PHP, musí poskytovat intuitivní ovládání všech dostupných funkcí a mít uživatelsky přívětivý vzhled.

Dále budou rozebrány návrhy implementace jednotlivých funkčních částí.

5.2 Návrh jednotlivých funkčních částí

Jelikož má být komponenta implementována v programovacím jazyce PHP, lze většinu funkcí implementovat přímo pomocí tohoto jazyka. Tento přístup má výhodu v jednoduché implementaci. Zároveň je však nepříjemný pro koncového uživatele. Vyžaduje totiž, aby při každém volání funkce, musel webový prohlížeč zavolat server a celá zobrazovaná stránka se následně načetla znovu. To v některých případech znemožňuje plynulost práce uživatele.

V současností je trendem řešit většinu operací na straně klienta pomocí technologie AJAX. Pomocí této technologie lze dosáhnout rychlejší a kvalitnější práce, není také tolik zatěžován komunikační kanál.

Při návrhu implementace jsem se rozhodl jít střední cestou mezi uvedenými předchozími dvěma přístupy. Protože by bylo obtížné posílat všechny informace o daných sloupcích tabulky na klienta a tam je uchovávat, rozhodl jsem se, že na straně klienta budou prováděny pouze kontroly

vstupních polí a zobrazování částí komponenty. Ostatní funkce komponenty budou implementovány na straně serveru.

5.2.1 Zobrazování tabulky

Vstupem komponenty bude vždy sql dotaz, podle kterého komponenta načte a zobrazí příslušnou tabulku. Sql dotaz je nutné zadávat v následujícím tvaru:

```
SELECT t1.sloupec1 [,t1.sloupec2]
FROM tabulka t1 [,tabulka2 t2]
[WHERE wherePodmínka]
[GROUP BY t1.sloupec1 [,t2.sloupec1]]
[HAVING havingPodmínka]
[ORDER BY t1.sloupec1 [,t2.sloupec1]]
```

Protože komponenta obecně může pracovat s více tabulkami, které mohou obsahovat stejný název sloupce jako jiná tabulka, je zapotřebí, při zadávání jména sloupce, uvést i do které tabulky sloupec patří. Pro specifikaci názvů sloupců lze použít i značku *, symbolizující, že uživatel chce načíst všechny příslušné sloupce. Vytvořená komponenta bude umožňovat zobrazovat i výsledek dotazu obsahujícího klausule WHERE, GROUP BY, HAVING a ORDER BY.

Po zadání dotazu uživatelem bude zavolána funkce, která podle klíčových slov rozdělí sql dotaz na několik částí, které reprezentují jednotlivé klausule sql dotazu. Dotaz bude rozdělen na části kvůli jednoduší modifikaci těchto částí. Například při následném volání funkce filtru se změní pouze klausule WHERE a dotaz se nad databází znovu vykoná.

Zároveň budou z databázového katalogu načteny potřebné informace o sloupcích a tabulkách. Budou nás zajímat především následující informace:

- jaké sloupce jsou primární klíče
- jaké sloupce jsou nenulové
- datový typ sloupce
- maximální délka sloupce

Všechny tyto zjištěné informace, včetně částí sql dotazu budou uloženy do superglobálního pole $_SESSION, odkud s nimi mohou pracovat další funkce komponenty. Tímto způsobem se budou uchovávat všechny parametry komponenty. Další variantou by bylo použití cookies, které je také vhodné pro uchování kontextu stránek. Rozhodl jsem se pro session, protože se jedná o propracovanější metodu udržování kontextu.

5.2.2 Stránkování

Pokud má tabulka velký počet řádků, zobrazí se, kvůli přehlednosti, jen její část. Zároveň budou uvedeny odkazy na další řádky tabulky. Řádky tabulky, které se mají zobrazit, budou specifikovány přímo pomocí klausule LIMIT v sql dotazu.

5.2.3 Zobrazování sloupců

Po zobrazení tabulky bude možné vybrat, které sloupce budou zobrazeny a které zůstanou skryty. Tato funkce bude realizována pomocí JavaScriptu na straně klienta. Primárně budou všechny sloupce zobrazené.

5.2.4 Filtry

U každého sloupce bude moci uživatel nastavit filtr pro zobrazování. K nastavení filtru bude sloužit vstupní formulář. Po odeslání formuláře na server bude upravena klausule WHERE dotazu sql a tabulka bude znovu načtena a zobrazena s danými filtry.

Komponenta bude poskytovat filtry pro číselné i řetězcové typy. Sloupce, které obsahují číselný datový typ, bude možné porovnávat se zadanou vstupní hodnotou pomocí následujících operátorů: <, <=, =, <>, >=, >. U sloupců, které obsahují řetězec, nás bude zajímat, jestli daný řetězec obsahuje vstupní podřetězec, zda na vstupní podřetězec řetězec v sloupci začíná či končí.

5.2.5 Řazení

Řazení tabulky podle daného sloupce bude prováděno na obdobném principu jako předchozí funkce. U každého sloupce tabulky bude zobrazen odkaz na seřazení tabulky podle daného sloupce. Řadit řádky tabulky bude možné sestupně i vzestupně a podle všech běžných datových typů, které může sloupec obsahovat. Řazení tabulky by bylo možné provádět i pomocí jazyka PHP. Nemusely by tak být z databáze načítány nová data. Na druhou stranu by byl více zatížen server, na kterém by řazení dat probíhalo. Rozhodl jsem se tedy data řadit přímo v databázi.

5.2.6 Přidávání, mazání a editace řádků

Aby bylo možné upravovat řádky tabulek databáze, je nutné vědět, který řádek se právě upravuje. Jako identifikaci řádku jsem zvolil hodnoty primárních klíčů všech tabulek, ze kterých jsou data čerpána. Tento přístup má jediný nedostatek, pokud je sloupec primárním klíčem, musí se zakázat změna jeho hodnoty, protože by se ztratil kontext o který řádek se jedná. Při vytváření a editaci řádků, bude komponenta pomocí JavaSciptu zajišťovat kontrolu vstupních formulářů.

5.2.7 Další funkce

Kromě základní funkčnosti bude komponenta poskytovat možnost změny názvu zobrazovaného sloupce, validaci všech vstupních polí odesílaných na server (délka, prázdnost, datový typ) a bude uživateli zobrazovat všechny sql dotazy, které se nad databází vykonávají.

5.2.8 Vzhled komponenty

Jak už bylo uvedeno výše. Hlavním požadavkem na vzhled komponenty je, aby uživateli umožňovala intuitivní práci. Kvůli přehlednosti, kromě již zmíněného stránkování, budou některé funkce komponenty zobrazeny až po jejich aktivaci. Primárně nebudou například zobrazeny formuláře pro filtry a výběr zobrazovaných sloupců.

6 Implementace komplementy

Tato kapitola popisuje samotnou implementaci komponenty. První část obsahuje rozbor implementačních prostředí, které jsem při implementaci použil. V další části je rozebráno jak jsou implementovány jednotlivé součásti komponenty. Na konci kapitoly jsou uvedeny další možná rozšíření.

6.1 Implementační prostředí

V této části budou podrobněji popsány programovací jazyky a prostředí, které budou využity při implementaci univerzálního tabulkového editoru. K implementaci systému budou využity následující prostředí:

- jazyk HTML s využitím kaskádových stylů CSS
- skriptovací jazyk PHP
- technologie AJAX (skriptovací jazyk JavaScript, DOM model)
- databáze Oracle

V následujícím textu budou výše uvedené prostředí podrobněji rozebrána.

6.1.1 HTML

HyperText Markup Language, označovaný zkratkou HTML, je značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na Internetu.

Jazyk je aplikací dříve vyvinutého rozsáhlého univerzálního značkovacího jazyka SGML (Standard Generalized Markup Language). Vývoj HTML byl ovlivněn vývojem webových prohlížečů, které zpětně ovlivňovaly definici jazyka. [12]

6.1.1.1 Vývoj jazyka

V roce 1989 spolupracovali Tim Berners-Lee a Robert Caillau na propojeném informačním systému pro CERN, výzkumné centrum fyziky poblíž Ženevy ve Švýcarsku. V té době se pro tvorbu dokumentů obvykle používaly jazyky TeX, PostScript a také SGML. Berners-Lee si uvědomoval, že potřebují něco jednoduššího a v roce 1990 byl tedy navržen jazyk HTML a protokol pro jeho přenos v počítačové síti – HTTP (HyperText Transfer Protocol – přenosový protokol hypertextu). Zároveň také Tim Berners-Lee napsal první webový prohlížeč, který nazval WorldWideWeb.

V roce 1991 CERN zprovoznil svůj web. Současně organizace NCSA (National Center for Supercomputer Applications) vybídla Marca Andreessena a Erica Binu k vytvoření prohlížeče

Mosaic. Ten vznikl v roce 1993 ve verzích pro počítače IBM PC a Macintosh a měl obrovský úspěch. Byl to první prohlížeč s grafickým uživatelským rozhraním. Následoval rychlý rozvoj webu, takže bylo nutné pro HTML definovat standardy. [12]

Verze jazyka: [12]

- Verze 0.9 - Byla vydána zhruba v roce 1991. Nepodporuje grafický režim (verze, kterou vytvořil Tim Berners-Lee).
- Verze 2.0 - Zachycuje stav jazyka v polovině roku 1994. Standard vydala komunita IETF (Internet Engineering Task Force). Je to první verze, která odpovídá syntaxi SGML. Přidává k původní specifikaci interaktivní formuláře a podporu grafiky.
- Verze 3.2 - Byla vydána 14. ledna 1997 a zachycuje stav jazyka v roce 1996. Připravovaná verze HTML 3.0 nebyla nikdy přijata jako standard, protože byla příliš složitá a žádná firma nebyla schopna naprogramovat její podporu. Standard už vydalo konsorcium W3C, stejně jako následující verze. Přidává k jazyku tabulky, zarovnávání textu a stylové elementy pro ovlivňování vzhledu.
- Verze 4.0 - Byla vydána 18. prosince 1997. Do specifikace jazyka přibyly nové prvky pro tvorbu tabulek, formulářů a nově byly standardizovány rámy (frames). Tato verze se snaží dosáhnout původního účelu – prvky by měly vyznačovat význam (sémantiku) jednotlivých částí dokumentu, vzhled má být ovlivňován připojovanými styly. Některé prezentační elementy byly zavrženy.
- Verze 4.01 - Byla vydána 24. prosince 1999. Tato verze opravuje některé chyby verze předchozí. Podle původního předpokladu se mělo jednat o poslední verzi, po které by se přešlo na XHTML.
- Verze 5 - 7. března 2007 byla založena nová pracovní skupina HTML, jejíž cílem je vývoj nové verze HTML. V květnu 2007 bylo odhlasováno, že základem nové specifikace se stanou Web Applications 1.0 a Web Forms 2.0 ze specifikace WHATWG. Jako název nové specifikace bylo odhlasováno HTML 5.0 a specifikace by měla být hotova v roce 2010 (odtud ji začnou vývojáři webových aplikací používat, finální verze zbavená všech chyb se však odhaduje až na rok 2022).

6.1.1.2 Přehled jazyka

Koncepce

Jazyk HTML je od verze 2.0 aplikací SGML. Je charakterizován množinou značek a jejich atributů, definovaných pro danou verzi. Mezi značky se uzavírají části textu dokumentu a tím se určuje význam (sémantika) obsaženého textu. Názvy jednotlivých značek se uzavírají mezi úhlové závorky (<a>). Část dokumentu tvořená otevírací značkou, nějakým obsahem a odpovídající ukončovací

značkou tvoří tzv. element (prvek) dokumentu. Například je otevírací značka pro zvýraznění textu a Červená Karkulka je element obsahující zvýrazněný text. Součástí obsahu elementu mohou být další vnořené elementy. Atributy jsou doplňující informace, které upřesňují vlastnosti elementu.
Značky (zvané tagy) jsou obvykle párové (v XHTML jsou párové všechny), přičemž koncové značka je shodná se značkou počáteční, jen má před názvem znak lomítko. Příklad pro označení odstavce:
Text odstavce

Některé značky jsou nepárové – nemají žádný obsah a nepoužívají koncovou značku. Příklad pro vykreslení vodorovné čáry:
<hr>

Tagy mohou obsahovat atributy, které popisují jejich vlastnosti nebo nesou jinou informaci. Příkladem může být odkaz (tag a), jehož atribut href říká, kam se uživatel po kliknutí na něj dostane (v tomto příkladu na stránku http://example.com):
text odkazu

Jelikož je HTML aplikací SGML, existují i jiné, méně známé konstrukce pro tvoření elementů. Jedná se o tzv. zkrácené HTML zápisy:

- <element/Obsah/ je totéž jako <element>Obsah</element>
- <element>Obsah</> je totéž jako <element>Obsah</element>
- <el<el2>Obsah je totéž jako <el><el2>Obsah
- Adam<>Božena je totéž jako Adam
Božena

Všechny tyto zápisy jsou sice podle normy validní a zcela ekvivalentní, ale žádný ze známých prohlížečů zkrácené verze nepodporuje, takže se nedoporučuje je používat.

Pro každou verzi existuje definice pravidel DTD (Document Type Definition). Od verze 4.01 musí být odkaz na deklaraci DTD v dokumentu uveden pomocí klíčového slova DOCTYPE. DTD definuje pro určitou verzi elementy a atributy, které lze používat. [12]

Dokument může mimo značkování obsahovat další prvky: [12]

- Direktivy – začínají znaky <!, jsou určeny pro zpracovatele dokumentu (prohlížeč).
- Komentáře – pomocné texty pro programátora, nejsou součástí obsahu dokumentu a nezobrazují se (prohlížeč je ignoruje).
- Kód skriptovacích jazyků.
- Definice událostí a kód pro jejich obsluhu.

Struktura jazyka

Dokument v jazyku HTML má předepsanou strukturu: [12]

- Deklarace DTD – je povinná až ve verzi 4.01, je uvedena direktivou <!DOCTYPE.
- Kořenový element – element html (značky <html> a </html>) reprezentuje celý dokument. Kořenový element je povinný, ale otevírací a ukončovací značka samotná povinná není (pokud tyto značky nebudou v těle dokumentu uvedeny, prohlížeč si je sám doplní podle kontextu).
- Hlavička elementu – obsahuje metadata, která se vztahují k celému dokumentu. Definují např. název dokumentu, jazyk, kódování, klíčová slova, popis, použitý styl zobrazení. Hlavička je uzavřena mezi značky <head> a </head>. Element head je opět povinný, ale jeho otevírací a koncová značka povinná není, prohlížeč ji sám doplní podle kontextu.
- Tělo dokumentu – obsahuje vlastní text dokumentu. Vymezuje se značkami <body> a </body>. Element body je povinný, ale jeho otevírací a koncová značka povinná není, prohlížeč ji sám doplní podle kontextu.

Příklad zdrojového kódu: [12]

Příklad HTML dokumentu ve verzi 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
    <html>
      <!-- toto je komentář -->
      <head>
        <title>Titulek stránky</title>
      </head>
      <!-- tělo dokumentu -->
      <body>
        <h1>Nadpis stránky</h1>
        <p>Toto je tělo dokumentu</p>
      </body>
    </html>
```

6.1.2 PHP

PHP (rekurzivní zkratka PHP: Hypertext Preprocessor, „PHP: Hypertextový preprocesor“, původně Personal Home Page) je skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek. Nejčastěji se začleňuje přímo do struktury jazyka HTML, XHTML či WML, což je velmi výhodné pro tvorbu webových aplikací. PHP lze ovšem také použít i k tvorbě konzolových a desktopových aplikací.

PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti. Syntaxe jazyka kombinuje hned několik programovacích jazyků (Perl, C, Pascal a Java). PHP je nezávislý na platformě, skripty fungují bez úprav na mnoha různých operačních systémech. Obsahuje rozsáhlé knihovny funkcí pro zpracování textu, grafiky, práci se soubory, přístup k většině databázových serverů (mj. MySQL, ODBC, Oracle, PostgreSQL, MSSQL), podporu celé řady internetových protokolů (HTTP, SMTP, SNMP, FTP, IMAP, POP3, LDAP, …)

PHP se stalo velmi oblíbeným především díky jednoduchosti použití a tomu, že kombinuje vlastnosti více programovacích jazyků a nechává tak vývojáři částečnou svobodu v syntaxi. V kombinaci s databázovým serverem (především s MySQL nebo PostgreSQL) a webovým serverem Apache je často využíván k tvorbě webových aplikací. Díky velmi častému nasazení na serverech se vžila zkratka LAMP – tedy spojení Linux, Apache, MySQL a PHP nebo Perl.

S verzí PHP 5 se výrazně zlepšil přístup k objektově orientovanému programování podobný Javě. [13]

6.1.2.1 Historie

Od roku 1994 je PHP jedním z nejpoužívanějších způsobů tvorby dynamicky generovaných WWW stránek. Jeho tvůrce Rasmus Lerdorf jej vytvořil pro svou osobní potřebu přepsáním z Perlu do jazyka C. Sada skriptů byla vydána ještě v témže roce pod názvem Personal Home Page Tools, zkráceně PHP.

V polovině roku se systém PHP spojil s programem Form Interpreter stejného autora. Tak vzniklo PHP/FI 2.0. Zeev Suraski a Andi Gutmans v roce 1997 přepsali parser a zformovali tak základ PHP3. Současně byl název změněn na dnešní podobu PHP Hypertext Preprocessor. PHP3 vyšlo v roce 1998, bylo rychlejší, obsahovalo více funkcí. Také běželo i pod operačním systémem Windows.

V roce 2000 vychází PHP verze 4, o čtyři roky později pak verze 5 s vylepšeným objektovým přístupem, podobným jazyku Java. [13]

6.1.2.2 Ukázka kódu [13]

Takto v PHP vypadá skript Hello world:

```
<?php
    echo "Ahoj, světe!";
?>
```

6.1.3 AJAX

AJAX (Asynchronous JavaScript and XML) je obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí, ale vyžadují

použití moderních webových prohlížečů. AJAX může být chápán jako vylepšený JavaScript, protože jeho podstata spočívá v tom. že JavaScriptu na straně klienta je umožněno volat na pozadí server a podle potřeby tak získávat potřebná data. Tímto způsobem je možné aktualizovat některé části stránky bez nutnosti opětovně načítat stránku. AJAX slouží pro dosažení rovnováhy mezi aktivitami serveru a klienta při provádění akcí požadovaných uživatelem

Tyto aplikace jsou vyvíjeny s využitím technologií:

- HTML (nebo XHTML) a CSS pro prezentaci informací.
- DOM a JavaScript pro zobrazování a dynamické změny prezentovaných informací.
- XMLHttpRequest pro asynchronní výměnu dat s webovým serverem (typicky je užíván formát XML, ale je možné použít libovolný jiný formát včetně HTML, prostého textu, JSON či EBML).

Podobně jako DHTML, LAMP nebo SPA, AJAX ve skutečnosti není konkrétní jednotlivá technologie, ale pojem označující použití několika technologií dohromady s určitým cílem. [14,15]

Z technologie AJAX jsem při implementaci použil JavaScript a DOM model, které jsou popsány dále. Přestože jsem nevyužil objektu XMLHttpRequest pro asynchronní komunikaci se serverem, rozhodl jsem se nastínit problematiku AJAXu jako celku a až poté se věnovat použitým částem – JavaScript a DOM model.

6.1.3.1 Historie

Termín AJAX se poprvé veřejně objevil v dubnu 2005 v článku Jesse James Garretta, nazvaném Ajax: A New Approach to Web Applications (Ajax: Nový přístup k webovým aplikacím). Myšlenky, na kterých je AJAX založen, jsou však výrazně starší: mezi začátky lze zařadit zavedení elementu IFRAME v Microsoft Internet Explorer 3.0 z roku 1996, elementu LAYER v Netscape Navigator 4.0 z roku 1997 (tento element byl opuštěn na počátku vývoje Mozilly). Také Macromedia Flash od verze 4 umožňoval komunikaci se serverem na pozadí, bez překreslení stránky.

V roce 1998 představil Microsoft novou technologii nazvanou Remote Scripting, ve které v klientském prohlížeči běžel Java applet komunikující se serverem, přičemž tento aplet poskytoval služby JavaScriptovým funkcím. Tato technika fungovala v MSIE od verze 4 i v Netscape Navigatoru od verze 4. V páté verzi IE zavedl Microsoft objekt XMLHttpRequest, který v roce 2000 využil v novém programu Outlook Web Access, který poskytuje webové rozhraní pro přístup k e-mailům na Microsoft Exchange Server.

Velká popularita a rozšíření AJAXu začala několika službami společnosti Google (nejdříve Gmail, posléze Google Maps a další). [14]

6.1.3.2 Výhody a nevýhody

Mezi výhody patří odstranění nutnosti znovunačtení a překreslení celé stránky při každé operaci, které jsou nutné u klasického modelu WWW stránek. Pokud například uživatel klikne na tlačítko pro udělení hlasu v nějaké anketě, celá stránka se musí znovu načíst ze serveru, třebaže se na ní jen například aktualizují výsledky hlasování a veškerý zbytek obsahu zůstává stejný. Prostřednictvím AJAXu proběhne odeslání hlasu uživatele na pozadí, server zašle jen ty části stránky, které se změnily, a jen tyto části se uživateli na stránce aktualizují a překreslí. Uživatel tak má pocit mnohem větší plynulosti práce, která se blíží běžným desktopovým aplikacím.

Z toho vyplývá také potenciál snížit zátěž na webové servery a síť obecně. Jelikož není potřeba při každém požadavku sestavit celý HTML dokument, ale pouze provedené změny, je množství vyměňovaných dat výrazně nižší a teoreticky to může mít příznivý vliv i na zátěž databázových serverů či dalších backendových systémů. AJAX však naopak může zvýšit počet vyměňovaných HTTP požadavků, třebaže přenášejí nižší množství dat, tak při nevhodné implementaci zátěž neklesne.

Mezi nevýhody patří hlavně změny v paradigmatu používání webu: webové stránky se chovají jako plnohodnotná aplikace se složitou vnitřní logikou, nikoli jako posloupnost stránek, mezi kterými se lze navigovat i pomocí tlačítek Zpět a Další. Moderní AJAXové aplikace jsou schopny funkce těchto tlačítek (přinejmenším částečně) obnovit za použití různých technik (např. využití části adresy za znakem # či pomocí neviditelných IFRAMEs).

Problémem AJAXových aplikací také může být síťová latence: potřeba komunikace přes Internet má negativní dopady na rychlost odezvy a interaktivitu uživatelského rozhraní. Pokud uživateli není jasně signalizováno, že aplikace zpracovává jeho požadavek (a na pozadí komunikuje se serverem), jediné, co zaregistruje, je zpožděná reakce (mezitím se dokonce může snažit operaci spustit znovu, neboť se domnívá, že systém jeho příkaz ignoroval).

Další nevýhodou AJAXu je nutnost používat moderní grafické prohlížeče, které podporují potřebné technologie. Všechny dnešní běžné prohlížeče však tyto technologie alespoň v základu podporují. [14]

6.1.3.3 Objekt XMLHttpRequest

Nejvýznamnějším stavebním kamenem interaktivní internetové aplikace využívající přístup AJAXu je objekt XMLHttpRequest. Prostřednictvím jeho vlastností můžeme sestavit požadavek HTTP na data potřebná pro aktualizaci a zaslat ho serveru asynchronně. Zatímco server zpracovává odpověď, klient může reagovat na podněty od uživatele a poskytnout mu komfort vyplývající z okamžité odezvy na jím generované události.

Objekt XMLHttpRequest byl vytvořen pro Internet Explorer jako prvek ActiveX již v roce 1999 firmou Microsoft. Vývojáři ostatních internetových prohlížečů naštěstí uznali, že se jedná o kvalitní nástroj, a tak je jeho použití možné ve všech současných moderních prohlížečích (Mozilla

FireFox, Opera). Tento fakt je velmi důležitý, protože objekt XMLHttpRequest není součástí standardu W3C. Způsoby jak tento objekt vytvořit se v různých typech prohlížečů liší, ale programové rozhraní API (Application Programming Interface) všech jeho instancí je shodný.

Nové prohlížeče, které přejímají technologii od Microsoftu, implementují XMLHttpRequest jako nativní objekt. Pro jeho konstrukci je možné použít následující příkaz:

```
var xmlHttp = new XMLHttpRequest();
```

V Internet Exploreru se objekt vytváří stejným způsobem jako ostatní členové rodiny ActiveX.

```
xmlHttp = new ActiveXObject('Microsoft.xmlHttp');
```
[17]

6.1.4 JavaScript

JavaScript je interpretovaný programovací jazyk pro WWW stránky, vkládaný přímo do HTML kódu stránky. Jeho syntaxe patří do rodiny jazyků C/C++/Java. Slovo Java je však součástí jeho názvu pouze s marketingových důvodů a s programovacím jazykem Java jej vedle názvu spojuje jen podobná syntaxe. Název jazyku používaný ve standardech je ECMAScript, JavaScript byl původně obchodní název implementace společnosti Netscape. Správný obchodní název pro verzi použitou v prohlížeči firmy Microsoft je JScript.

JavaScript je vkládán do WWW stránek na Internetu, které jsou samy o sobě tvořeny kódem HTML. Při prohlížení těchto stránek JavaScript stažen spolu s WWW stránkou spuštěn ve webovém prohlížeči. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

Program v JavaScriptu se spouští až po stažení z Internetu (tzv. na straně klienta), narozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele. [16]

6.1.5 Document Object Model (DOM)

Webová stránka, která se zobrazuje v klientském prohlížeči je dokument, jehož struktura je definovaná pomocí značkovacího jazyka HTML. Při interpretaci zdrojového kódu stránky klientským prohlížečem vzniká také objektový model dokumentu DOM. Pomocí programu v JavaScriptu, který může být součástí stránky, je možné s tímto modelem pracovat a měnit tak vzhled a obsah dokumentu.

DOM je tvořen množinou objektů reprezentujících jednotlivé elementy dokumentu. Tyto objekty vytvářejí strom dokumentu, jehož vrcholem je objekt reprezentující samotný dokument. DOM model také umožňuje vytvářet dynamicky nové uzly a libovolný uzel zrušit, přičemž rušení uzlu má za následek zrušení všech jeho dětí.

DOM přestavuje silný nástroj pro práci s obsahem dokumentu a dovoluje poměrně významné změny, které jsou rychlé a nevyžadují volání serveru. Práce s DOM tedy patří do skupiny nástrojů využívaných AJAXem. [17]

6.1.5.1 Základní vlastnosti a metody objektů DOM modelu

V této části budou uvedeny základní vlastnosti a metody objektů DOM modelu, které jsou důležité pro základní operace nad stromem dokumentu.

Vlastnost	**Popis**
Základní vlastnosti	
nodeType	Každý uzel stromu dokumentu má svůj typ. Existuje celkem 12 typů podle W3C standardu. Nejčastější dva z nich jsou ELEMENT_NODE, reprezentující libovolný prvek dokumentu (odstavec, obrázek, odkaz, nadpis, seznam,...), TEXT_NODE, představující fragment textu. Tento uzel již nemá žádné potomky.
id	Jedinečný identifikátor prostřednictvím něhož lze získat ukazatel na příslušný uzel/prvek.
className	Uzlu/prvku specifikovat třídu, do které patří.
tagName	Název značky, kterou je v HTML příslušný prvek označen.
style	Zpřístupní množinu vlastností souvisejících se vzhledem prvku. Názvy dílčích vlastností jsou až na malé odlišnosti shodné s CSS vlastnostmi.
Vlastnosti pro pohyb ve stromu	
childNodes	Pole ukazatelů na nejbližší potomky.
parentNodes	Ukazatel na rodiče.
firstChild	Ukazatel na prvního nejbližšího potomka.
lastChild	Ukazatel na posledního nejbližšího potomka.
nextSibling	Ukazatel na následující uzel z rodičovského seznamu potomků.
previousSibling	Ukazatel na předcházející uzel z rodičovského seznamu potomků.

Tabulka 6.1: Základní vlastnosti objektů DOM modelu [17]

Metoda	**Popis**
element. getElementById(id)	Vrátí ukazatel na prvek s příslušným id. Nejrychlejší způsob, jak se dostat k příslušnému uzlu/prvku.
element. getElementsByTagName(tagName)	Rekurzivně prohledává strom od uzlu, na kterém byla metoda zavolána, směrem dolů a vrací pole ukazatelů na uzly, jejichž tagName se shoduje s parametrem metody.

element. createElement(tagName)	Vytvoří objekt reprezentující nový prvek podle názvu značky tagName a vrátí na něj ukazatel.
element. createTextNode(text)	Vytvoří objekt reprezentující textový uzel, naplní ho textem podle parametru a vrátí na něj ukazatel.
element. appendChild(child_ptr)	Připojí uzel do seznamu dětí na poslední místo.
element. removeChild(child_ptr)	Odstraní uzel ze seznamu dětí.

Tabulka 6.2: Základní metody objektů DOM modelu [17]

Pomocí těchto vlastností a metod již není problém uchopit libovolný prvek dokumentu a modifikovat jeho vlastnosti nebo přidat či naopak zrušit jiný prvek. [17]

6.1.6 Oracle

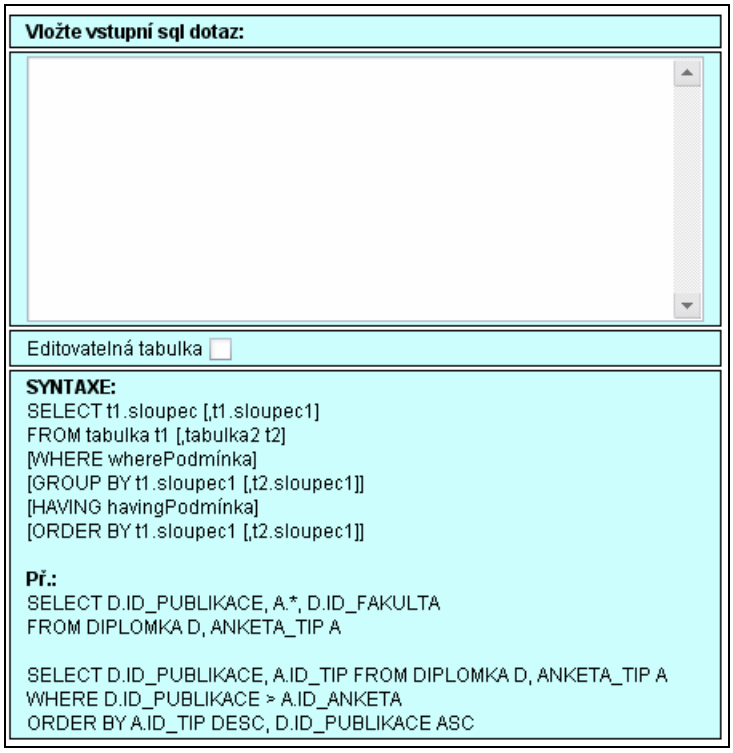
Oracle umožňuje ukládat data a získávat přístup k nim způsobem odpovídajícím definovanému modelu, které je znám jako relační model. Z tohoto důvodu je Oracle považován za systém správy relační databáze, či chcete-li za relační systém řízení báze dat. Kromě uložení dat v relačním formátu podporuje Oracle objektově orientované struktury, například abstraktní datové typy a metody. Objekty lze provázat s jinými objekty nebo také mohou obsahovat jiné objekty.

Bez ohledu na to, zda použijeme relační nebo objektově orientované struktury, ukládá server Oracle data do souborů. Data jsou interními strukturami databáze logicky mapována na soubory. Různé typy dat lze tedy ukládat samostatně. Tato logická rozdělení se nazývají tabulkové prostory (každá databáze obsahuje alespoň jeden tabulkový prostor).

Aktuální verzí je Oracle Database 11g. Tento systém podporuje nejen standardní relační dotazovací jazyk SQL podle normy SQL92, ale také proprietární firemní rozšíření Oracle (např. pro hierarchické dotazy), imperativní programovací jazyk PL/SQL rozšiřující možnosti vlastního SQL (v tomto jazyce je možné tvořit uložené procedury, uživatelské funkce, programové balíky a triggery), dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat (XML databáze, jazyk XSQL). [18,19]

6.2 Implementace komponenty

Komponenta, po zadání dotazu do vstupního formuláře (obr.6.1), použije pro získání vnitřní interpretace dotazu třídu Parser. Tato třída má jako parametr konstruktoru právě sql dotaz, který rozdělí na jednotlivé klausule. Tyto klausule jsou poté uloženy do superglobálního pole $_SESSION, odkud s nimi mohou pracovat další funkce komponenty. Tímto způsobem se ukládají všechny informace, které jsou nutné k uchování stavu komponenty.

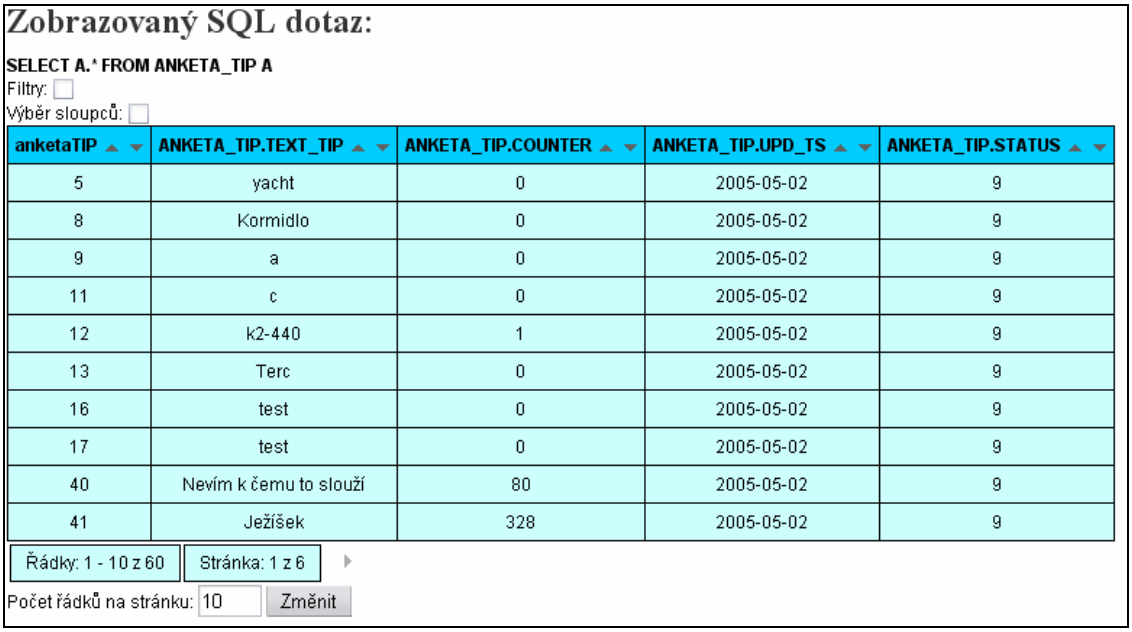


Obrázek 6.1: Vstupní formulář pro vkládání sql dotazu

Následně je volána funkce getInfoFromDatabase($sql, $sloupec, $tabulka, $vsechnySloupce), která zajistí načtení informací z databázového katalogu. První parametrem je sql dotaz, druhým pole všech sloupců, třetím pole tabulek a poslední parametr označuje, zda se mají zobrazit všechny sloupce tabulky (v tomto případě funkce nejprve načte sloupce z databáze a až poté zjistí jejich vlastnosti). Funkce vrací pole všech sloupců, ve kterém jsou zjištěné vlastnosti sloupců uvedeny.

Dále je volána funkce setZnaceniSloupcu($sloupec), která má jako jediný parametr pole sloupců. Tato funkce obsahuje převodní pole pro názvy sloupců. Názvy sloupců tak nemusí být zobrazovány přímo jak jsou uloženy v databázi, ale uživatel si je může podle své potřeby upravit, což také zvyšuje přehlednost tabulky.

Když máme všechny potřebné informace, zavolá se nad třídou Table funkce, která sestaví výslednou tabulku (získá jí podle sql dotazu z databáze). Poté se zavolají funkce printFilters ($sloupec), printColumnSelect ($sloupec), printTable ($sloupec), printPaging ($rowFrom,$rowCount,$rowCountTotal), které vytisknou tabulku a další funkční části komponenty. Obrázek 6.2 ukazuje jak zobrazená tabulka vypadá.



Obrázek 6.2: Tabulka zobrazená podle vstupního sql dotazu

6.2.1 Stránkování

Pro přechody mezi jednotlivými částmi tabulky jsou zobrazeny odkazy, podle kterých lze identifikovat, které řádky se mají načíst. Pokud uživatel na tento odkaz klikne, zavolá se server, který zjistí jaké řádky se mají zobrazit a pomocí klausule LIMIT sql dotazu tyto řádky načte z databáze. Následně zobrazí novou tabulku. Zároveň se kontroluje, jestli dané řádky skutečně existují. Obdobným způsobem lze také nastavit počet zobrazovaných řádků na stránku.

Při zobrazení tabulky nechybí informace o tom, které řádky se zobrazují. Pokud neexistují žádné řádky před již zobrazovanými, odkaz na předchozí stránku se nezobrazí. Obdobně i na konci tabulky.

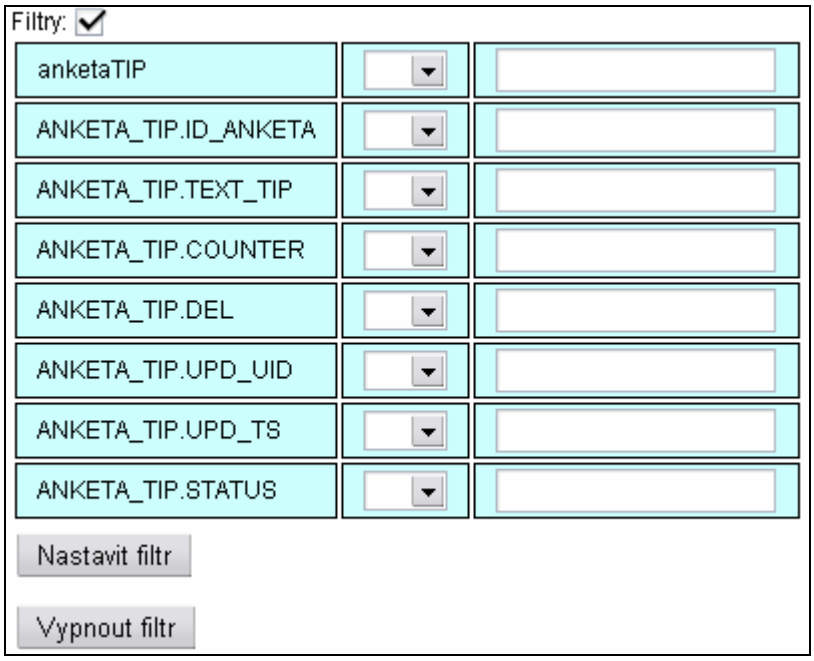
6.2.2 Řazení

Komponenta umožňuje řazení podle různých sloupců. U každého sloupce jsou vygenerovány odkazy pro sestupné a vzestupné řazení. Pokud uživatel na některý z těchto odkazů klikne, zavolá se server, který zjistí podle kterého sloupce chce uživatel tabulku seřadit, upraví klausuli ORDER BY, načte data z databáze a klientovy pošle novou tabulku.

6.2.3 Filtry

Použití filtrů je nastavováno pomocí vstupního formuláře. Pomocí javascriptu je při odesílání formuláře provedena kontrola, zda jsou vstupní data ve správném tvaru. Kontroluje se jestli vstup

vyhovuje datovému typu daného sloupce, délka vstupu a vyplnění vztahu. V případě špatného zadání filtru je na tuto skutečnost uživatel upozorněn a vyzván aby filtr opravil.

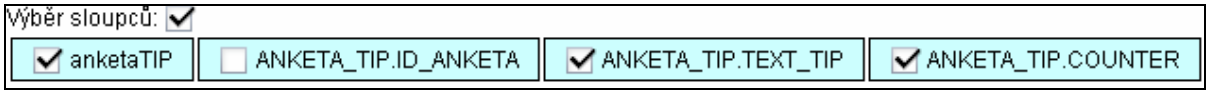


Obrázek 6.3: Pole pro výběr zobrazovaných sloupců

Po odeslání vstupního formuláře server zpracuje odeslaná data, nastaví klausuli WHERE sql dotazu. Obdobně jako u ostatních funkcí jsou z databáze načtena nová data a ty poslána na klienta. Při aplikaci filtru nad sloupcem, který obsahuje řetězce, je použita klausule LIKE.

6.2.4 Zobrazování sloupců

Po načtení tabulky může uživatel specifikovat, které sloupce zobrazovat a které skrýt (viz. obr.6.3).



Obrázek 6.4: Pole pro výběr zobrazovaných sloupců

Tato funkce je zajišťována pomocí JavaScriptu. Každá buňka tabulky má třídu nastavenou na jméno sloupce, ke kterému patří. Podle této třídy je identifikovatelná. Pro získání buněk daného sloupce se používá následující funkce, která vrátí pole všech objektů s danou třídou.

```
function getElementsByClassName(strClass, strTag, objContElm)  {
        strTag = strTag || "*";
        objContElm = objContElm || document;
        var objColl = objContElm.getElementsByTagName(strTag);
        if (!objColl.length &&  strTag == "*" &&  objContElm.all) objColl = objContElm.all;
        var arr = new Array();
        var delim = strClass.indexOf('|') != -1  ? '|' : ' ';
        var arrClass = strClass.split(delim);
        for (var i = 0, j = objColl.length; i < j; i++) {
          var arrObjClass = objColl[i].className.split(' ');
          if (delim == ' ' && arrClass.length > arrObjClass.length) continue;
          var c = 0;
          comparisonLoop:
          for (var k = 0, l = arrObjClass.length; k < l; k++) {
            for (var m = 0, n = arrClass.length; m < n; m++) {
              if (arrClass[m] == arrObjClass[k]) c++;
              if (( delim == '|' && c == 1) || (delim == ' ' && c == arrClass.length)) {
                arr.push(objColl[i]);
                break comparisonLoop;
              }
            }
          }
        }
        return arr;
      }
```

V případě skrývání sloupce se všem takto získaným buňkám nastaví vlastnost display na 'none'. Pokud chce uživatel znovu sloupec zobrazit, vlastnost display se u všech buněk sloupce nastaví zpět na předchozí tvar.

6.2.5 Přidávání, mazání a editace řádků

Komponenta zajišťuje modifikaci tabulky následujícím způsobem. Každý řádek je identifikovatelný podle hodnot všech primárních klíčů. Zobrazovaná tabulka se může skládat z více spojených databázových tabulek. Pokud tomu tak je, úpravy se promítnou ve všech databázových tabulkách ze kterých se skládá.

Při mazání řádků tabulky se pouze odešle příkaz na server a ten podle identifikace řádku smaže příslušné řádky tabulek v databázi. Pro kontrolu je zobrazen vykonaný sql dotaz. Následně je zobrazena již takto modifikovaná tabulka.

Pro editaci a vkládání řádků je v tabulce k dispozici formulář, který je při načtení stránky skrytý. Formulář se zobrazí až aktivaci příslušné funkce. Při odesílání formuláře jsou kontrolována všechna odesílaná data. Testuje se délka dat, typ dat a v případě, že daný sloupec musí být nenulový se testuje i to, že není zadána prázdná položka. Po odeslání formuláře server zajistí provedení změn v databázi. Opět je pro kontrolu zobrazen dotaz, který se provedl.

Na následujícím obrázku je zobrazen formulář, pomocí kterého se provádí editace daného řádku. Jak je z obrázku patrné, sloupec anketaTip nelze editovat. Tento sloupec jo totiž primárním klíčem a jeho editace je zakázaná z důvodu udržení kontextu (o který řádek se jedná).

anketaTIP	ANKETA_TIP.ID_ANKETA	ANKETA_TIP.TEXT_TIP	ANKETA_TIP.UPD_TS	ANKETA_TIP.STATUS	
5	1	yacht	2005-05-02	9	
8	1	Kormidlo	2005-05-02	9	

Obrázek 6.5: Editace položek

6.3 Další rozvoj systému

Tato kapitola diskutuje další možnosti rozvoje vytvořené komponenty. Jednotlivá rozšíření by měla vést k celkovému usnadnění práce s touto komponentou.

6.3.1 Použití AJAXu pro odesílání formulářů

Rychlejší práci s komponentou by jistě zajistilo, kdyby se i některé další implementace funkcí komponenty přesunuly na stranu klienta. Například při odesílání dat na server by šlo využít objekt XMLHttpRequest, jak je popsáno v teoretickém rozboru jednotlivých implementačních technologií v části týkající se technologie AJAX.

6.3.2 Kontrola primárních a cizích klíčů

Komponenta pomocí Javascriptu a DOM modelu kontroluje správnost všech odesílaných dat na server. Kontroluje se délka vstupu, správný datový typ, případná nenulovost vstupního pole. Dalším možným rozšířením by bylo zavedení kontroly primárních a cizích klíčů přímo na straně klienta. Při vkládání nového řádku tabulky by komponenta kontrolovala, zda hodnota sloupce, který je primárním klíčem, se již v databázi nevyskytuje, a jestli odkazovaná hodnota sloupce, který je cizím klíčem, se v

databázi naopak vyskytuje. U sloupců, které jsou cizími klíči, by případně komponenta místo vstupního pole pro vkládání řetězce mohla poskytovat výběrové pole, které by poskytovalo seznam všech hodnot, kterých sloupec může nabývat (podle daného integritního omezení).

7 Závěr

Jak už bylo napsáno v úvodu, cílem této diplomové práce je vytvořit univerzální tabulkový editor. Výsledkem je komponenta, která uživateli usnadní práci s databázovým systémem Oracle. Tato komponenta byla vytvořena pomocí programovacího jazyka PHP a bude sloužit pro pracovníky Centra výpočetních a informačních služeb VUT v Brně. Pro jednoduchost následné modifikace zdrojového kódu jsem využil u některých částí implementace objektových vlastností jazyka PHP5.

Největší důraz při implementaci této komponenty byl kladen na to, aby byla práce s komponentou intuitivní a uživatelsky přívětivá. Komponenta umožňuje pracovat se všemi daty uloženými v databázi Oracle. Tabulka s daty je zobrazena na základě zadaného vstupního sql dotazu.

Komponenta umožňuje řazení řádků tabulky podle různých sloupců, listování v tabulce, která má mnoho položek, skrývání a zobrazování sloupců tabulky. Při zobrazování poskytuje možnost použití filtru, možnost měnit jména zobrazovaných sloupců tabulek a přímou editaci dat v tabulce.

Při implementaci jednotlivých funkcí jsem především vycházel z konkrétních požadavků pracovníků CVIS VUT v Brně a tabulkový editor přizpůsoboval jejich potřebám.

Pro zmenšení zatížení databázového serveru a komunikace na síti jsou některé funkce komponenty implementovány přímo na straně klienta pomocí JavaScriptu a DOM modelu.

Jediným omezením této komponenty je nefunkčnost některých částí při použití v některých verzích prohlížeče Mozilla Firefox. Důvodem je, že tento prohlížeč netradičně interpretuje procházení DOM modelem. Prázdné znaky (mezera, nová řádek) mezi HTML značkami prohlížeč klasifikuje jako uzly DOM modelu, tím je znemožněno klasické procházení DOM modelem. Pracovníci CVIS VUT v Brně byli s tímto nedostatkem seznámeni. V ostatních prohlížečích se chová komponenta standardně.

Vytvořená komponenta poskytuje všechny funkce, které jsou specifikovány v zadání této diplomové práce. Součástí této práce je i návrh dalších možných rozšíření.

Literatura

[1] *Databáze* [online]. [cit. 2007-12-19].
Dostupné na URL: <http://cs.wikipedia.org/wiki/Databáze>.

[2] POKORNÝ, J.: *Dotazovací jazyky*. 1994. 227 s. ISBN 80-901475-2-6.

[3] *Relační vs. objektově-relační vs. objektové databáze.* [online]. [cit. 2007-12-22].
Dostupné na URL: <http://www.fi.muni.cz/~xbatko/oracle/compare.html>.

[4] ZENDULKA, J., RUDOLFOVÁ, I.: *Databázové systémy*.
Skripta k předmětu Databázové systémy.

[5] *Relační algebra.* [online]. [cit. 2007-12-20].
Dostupné na URL: <http://homen.vsb.cz/~s1i95/ISVDAS/IS/IS_relace.htm>.

[6] *Databáze a jazyk SQL*. [online]. [cit. 2008-01-02].
Dostupné na URL: <http://interval.cz/clanky/databaze-a-jazyk-sql/>.

[7] *SQL*. [online]. [cit. 2007-01-02].
Dostupné na URL: <http://cs.wikipedia.org/wiki/SQL>.

[8] KOSEK, J.: *Přes W@P do databáze.* [online]. [cit. 2007-01-02].
Dostupné na URL <http://www.kosek.cz/clanky/wapkurz/ar06s48.html>.

[9] Strakoš, M.: *Vysokoškolský informační portál.* [diplomová práce].
Fakulta informačních technologií VUT v Brně.

[10] *phpMyAdmin*. [online]. [cit. 2008-05-01].
Dostupné na URL: <http://cs.wikipedia.org/wiki/PhpMyAdmin>.

[11] *phpPgAdmin*. [online]. [cit. 2008-05-01].
Dostupné na URL: <http://cs.wikipedia.org/wiki/PhpPgAdmin>.

[12] *HTML* [online]. [cit. 2007-12-22].
Dostupné na URL: <http://cs.wikipedia.org/wiki/HTML>.

[13] *PHP* [online]. [cit. 2007-12-22].
Dostupné na URL: <http://cs.wikipedia.org/wiki/PHP>.

[14] *AJAX* [online]. [cit. 2007-12-22].
Dostupné na URL: <http://cs.wikipedia.org/wiki/AJAX>.

[15] DARIE, C.: *Ajax a PHP*. Zoner Press; 2006; ISBN 80-86815-47-1

[16] *JavaScript* [online]. [cit. 2007-12-23].
Dostupné na URL: <http://cs.wikipedia.org/wiki/JavaScript>.

[17] MÁČEL, L., KUŽELA, A., HRUŠKA, T.: *Internetové aplikace (WAP) V. – část AJAX*.
Skripta k předmětu Internetové aplikace.

[18] *Oracle* [online]. [cit. 2007-12-22].
Dostupné na URL: <http://cs.wikipedia.org/wiki/Oracle>.

[19] LONEY, K., THERIAULT, M.: *Mistrovství v Oracle*. Oracle Press; 2002; ISBN 80-7226-635-7